Explaining Machine Learning Models by Generating Counterfactuals

Ivan Afonichkin

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 29.07.2019

Thesis supervisor:

Prof. Aristides Gionis

Aalto University
School of Science

AALTO UNIVERSITY
SCHOOL OF SCIENCE

ABSTRACT OF THE
MASTER'S THESIS

Author: Ivan Afonichkin

Title: Explaining Machine Learning Models by Generating Counterfactuals

Date: 29.07.2019 Language: English Number of pages: 4+47

Master's Programme in Computer, Communication and Information Sciences

Professorship: Machine Learning and Data Mining (Macadamia)

Supervisor and advisor: Prof. Aristides Gionis

Nowadays, machine learning is being applied in various domains, including safety critical areas, which directly affect our lives. These systems are so complex and rely on huge amounts of training data, so that we risk to create systems that we do not understand, which might lead to undesired behavior, such as fatal decisions, discrimination, ethnic bias, racism and others. Moreover, European Union recently adopted General Data Protection Regulation (GDPR), which requires companies to provide meaningful explanation of the logic behind decisions made by machine learning systems, if these decisions affect directly a human being.

We address the issue of explaining various machine-learning models by generating counterfactuals for given data points. Counterfactual is a transformation, which shows how to alternate an input object, so that a classifier predicts a different class. Counterfactuals allow us to better understand why particular classification decisions take place. They may aid in troubleshooting a classifier and identifying biases by looking at alternations needed to be made in the data instances. For example, if a loan approval application system denies a loan for a particular person, and we can find a counterfactual indicating that we need to change the gender, or the race of a person for the loan to be approved, then we have identified bias in the model and we need to study our classifier better and retrain it to avoid such undesired behavior.

In this thesis we propose a new framework to generate counterfactuals for a set of data points. The proposed framework aims to find a set of similar transformations to data points, such that those changes significantly reduce the probabilities of the target class. We argue that finding similar transformations for a set of data points helps to achieve more robust explanations to classifiers. We demonstrate our framework on 3 types of data: tabular, images and texts. We evaluate our model on both simple and real-world datasets, including ImageNet and 20 NewsGroups.

Keywords: machine learning, interpretability, counterfactuals, transparency

Contents

1 Introduction

Nowadays, powerful black-box systems are becoming more widespread. Machine learning is being applied in various domains concerning our day-to-day life including advertisements, recommendation systems, and voice recognition systems. Companies advertise their products through extensive use of sophisticated algorithms. A recent report by McKinsey [19] affirms that machine learning is going to be the next frontier for innovation.

1.1 The need for transparent systems

Black-box systems are used in safety-critical areas which directly affect our lives. Such areas include medicine, self-driving cars, criminal justice systems and automatic hiring systems. These systems are complex and rely on huge amounts of the training data, so we risk creating systems that we do not really understand. This might lead to undesired behaviours, such as fatal decisions, discrimination, ethnic bias, racism, and others.

The COMPAS system (machine learning system that predicts future criminals) has been criticized for being highly biased against black people [2]. This system evaluates black people twice as likely as white people to re-offend.

A study at Princeton by Caliskan et al. [7] shows how web corpora could be biased against race. In their corpora, many more unpleasant words are associated with black people's names than with those of white people. Such biased data can have undesirable implications for many machine learning tasks, e.g. sentiment analysis.

A study by Freitas [16] shows how accidental artifacts can result in very poor model performance. They report the USA military trained a classifier to distinguish between enemy and friendly tanks. The classifier had very good training dataset accuracy, but poor performance in the actual field. It was discovered that pictures of all friendly tanks were taken on sunny days, so the model just learned that if it is a sunny photo, it is a friendly tank.

1.2 Adversarial Examples

Deep neural networks are highly expressive and powerful models that have achieved state of the art results on many real-world problems, including speech recognition [18] and computer vision [22]. However, almost all highly expressive machine learning models are vulnerable to minimal changes to the input, which can drastically change output of the model [35]. In this work, authors have found that they can cause deep neural network to misclassify an image by adding small, almost invisible to a human eye, perturbation. *Adversarial examples* are called the objects, which obtained by applying these small perturbation to the objects. These small invisible perturbations are usually called *adversarial perturbations*. Example of an adversarial example can be found in Fig 1.

Authors have also discovered that the nature of this perturbation is not random, the same perturbation can be applied to different networks and will cause an object

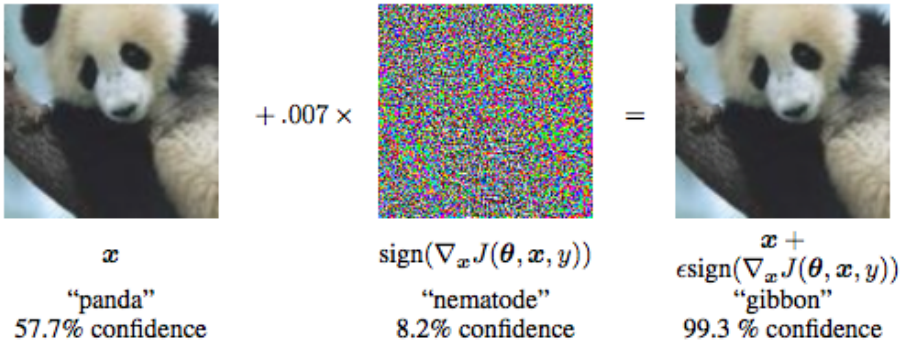


Figure 1: Adversarial example, which obtained by applying small, almost invisible, perturbation to the input image. As a result, network misclassified the object.

to be misclassified.

Algorithms for finding adversarial perturbations are called *adversarial attacks*. These algorithms have drawn a lot of attention, because they reveal vulnerabilities of machine learning models and are major threat to the security of systems utilizing machine learning models. As an example, minimal perturbations can turn any traffic sign into no speed limit sign, which can result in disaster.

1.3 General Data Protection Regulation (GDPR)

Recently, European Union adopted a new law called *General Data Protection Regulation (GDPR)*. GDPR gives more control to the citizens of EU over their personal data. One aspect of GDPR, which affects companies utilizing machine learning algorithms, is so called 'right for explanation'. It means that whenever an automated decision takes a place, every individual has a right to have 'meaningful explanation of the logic behind this particular decision made by the algorithm'. It applies certain restrictions to the algorithms used by the companies, because now companies need to be able to retrieve 'meaningful explanation' from every decision made that affects an individual.

1.4 Trust in the model

If there is an action that will be taken based on the prediction of the model or the model should be deployed 'into the wild', *trust* is usually a very important aspect. LIME paper [28] provides two definitions of *trust*:

1. *Trust in the model* means we generally trust the model and expect it to behave reasonably, when deployed 'into the wild'. Usually it might involve checking that real-world data distribution is very close to what has been used for training/validation, making sure model does not discriminate against any protected variables (gender/race), etc.

2. *Trust the prediction* means that we are confident enough in the prediction made by model, so that we can take an action based on it. For some applications decision based on the prediction of the model might be critical, e.g. in medical and military applications, so we should be very careful and confident about prediction in such applications, otherwise consequences can be disastrous.

1.5 Our contributions

In this thesis we propose a new framework to generate counterfactuals for a set of data points. Proposed framework is different from general counterfactual generation methods [15, 38, 31, 33, 28], because it provides an explanation for the whole target class, instead of an explanation for the specific data point. Framework provides an explanation for the whole class by finding a set of similar transformations to the given data points, such that probabilities of the target class for the modified data points significantly drop. We argue that these similar transformations serve as an explanation for the target class, because applying them to any data point in the dataset reduces the probability of the target class.

Our main contributions:

1. Proposed framework to generate counterfactuals for a set of data points.

2. Concrete settings of the proposed framework for three types of data: tabular, images, texts.

3. Experiments, which demonstrate robustness of the framework and consistency among explanations.

2 Background

In section 1 we have seen that having an understanding of how the model works might be crucial, especially in the setting when there is a decision to be made based on the output of the model, e.g. in medical diagnosis application Caruana et al. [8]. However, it is not always desirable or needed. If there is no decision to be made based on the output of machine learning algorithm, or the decision that will be made does not have any consequences, then there is no need to have interpretable model, in this case any powerful model will work. In further text we consider the case, when interpretability of the model is highly desirable.

Before discussing interpretability, it is important to define what it means and understand different aspects of interpretable models. According to dictionary[1], *interpretability* means the ability to tell or explain some concept in understandable terms. This definition involves several important aspects. First, it is subjective to a particular person, because of the part *'in understandable terms'*. Different people might have different backgrounds and explaining how particular machine learning model works really depends on the person. Second, this definition involves *'concept'*, which should be well-defined in order to be able to explain it clearly. In this thesis, we will refer to *interpretability* as ability to provide *explanations*, which we define in the next section 2.1.

2.1 What is an explanation?

In this thesis, we refer to an *explanation* as either textual or visual representation, which helps to understand the relationship between the input's components and model's output. This definition of *explanation* is inherently ambiguous, because it mentions 'representation, which helps to understand...', which is subjective in its nature. There might be a huge discussion regarding which explanations are better to what kind of people, however we leave this discussion out of this thesis and concentrate on, as we think, more practical aspects.

Basically, we refer to an *explanation* as anything that helps us to better understand the relationship between the model's input components and its output. To better understand it, there are few examples of explanations used through this text:

1. One way to generate explanations for *linear models* 2.3.1 is by visualizing weights as in figure 2. This visual representation clearly establishes relationship between input components (features) and model's output. Magnitude of the weights allow us to reason which features have bigger impact on the output, while sign of the weight tells us whether increasing value of the feature affect output in an increasing or decreasing way.

2. It is common to provide explanations for *decision trees* 2.3.2 just by visualizing them as in figure 3. This type of explanation clearly shows the relationship between input and model's output just by following every path from the root node to one of its leaves.

[1] https://www.merriam-webster.com/

Explainer is a method, which provides explanations for a given machine learning model. For inherently interpretable models such as *linear models* 2.3.1 and *decision trees* 2.3.2, there is no need to have external explainers, because they are explainers by themselves. For some models, such as Deep Neural Networks [22], there is usually a need to have an external explainer.

2.2 Aspects of interpretability

According to Guidotti et al. [17], there are several aspects of interpretability: *global and local explainers*, *time limitation*, *user expertise*.

2.2.1 Global vs Local explainers

Explainer is called *global*, if it provides explanations, which help us to fully understand the entire reasoning that leads to different outcomes for a given model. In the case when we have a global explainer for a given model, it is possible to understand the underlying logic of particular model. Few examples of models, which are *global explainers* by themselves: linear regression, decision trees. We call such models *globally interpretable*.

Explainer is called *local*, if it provides explanations, which only help us to understand reasons for particular model's output on a given input. Methods that fall into this category usually give explanations of why particular decision was made only for single datapoint. Recent method that has received considerable attention and belongs to this category is LIME [28].

To see the better contrast between *global* and *local* explainers, consider an example of model, which takes image as an input and outputs most probable class. If our model receives image of a cat as an input, and classifies this image as a 'cat', *local explainer* might provide an explanation in form of 'This image has been classified as a cat, because it has whiskers'. While in the case of *global explainer*, we can get an explanation, which clearly states what needs to be in the image in order for it to be classified as any of the available classes, so it is not local and not specific to any particular input.

2.2.2 Time Limitation

Time Limitation of the user to understand provided explanation is an important aspect. This aspect is usually related to the case, when output of the algorithm has to be used in order to make a decision.

Sometimes, provided explanations could be not feasible to understand in timely manner. For example, popular method to explain linear regression model is to look at the learned weights as in figure 2, however if the model has learned more than hundreds of thousands weights, it might be not feasible to understand the model in timely manner. That is why sparse explanations (small number of weights) are usually preferred as explanations.

Another example is decision tree. People usually agree that decision trees are interpretable and easy to understand (it is easy to reason why particular decision

about input datapoint was made), however if we have very big decision tree with very high depth, it might be not feasible to understand the logic behind this tree. As a general rule, shorter and more concise explanations are preferred.

Usually, time required to understand an explanation is approximated using complexity of the model. For example, in the case of decision trees it could be depth of the tree or number of leaves. In the case of linear regression models it is usually number of non-zero parameters (that is why sparse models are preferred). Then, on the stage of generating an explanation, this proxy for required time is included in the cost function, such that there is a trade-off between complexity of explanation and its faithfulness.

2.2.3 User Expertise

Provided explanations for a given model can have different users, e.g. people who make decisions based on the output of model, data scientists, software engineers, etc. As it already has been mentioned in the beginning of section 2, different people might have different backgrounds, thus might require different explanations. For example, experts in particular field might prefer more complex explanations, which highlight different aspects of the underlying phenomena, while other people prefer the simplest explanations possible. As mentioned in 2.1, we leave out the discussion of user expertise from this thesis and, as we think, concentrate on more practical aspects.

2.3 Globally Interpretable Models

As we have already mentioned in 2.2.1, model is globally interpretable, when we can fully understand the entire reasoning that leads to different outcomes. There are several inherently globally interpretable models that are usually recognized in traditional machine learning: *linear models, decision trees, rules and lists.*

2.3.1 Linear models

Linear models are the models, which approximate relationship between input and output variables by using linear predictor functions, and estimate parameters of the model from available training data. Given a sample of input data points $X_1, X_2, ..., X_n$, where each $X_i = (X_{i1}, X_{i2}, ..., X_{ik})$ with corresponding labels $Y_1, Y_2, ..., Y_n$, the relationship with output variables is modeled as follows:

$$Y_i = g(w_0 + w_1 \cdot X_{i1} + ... + w_k \cdot X_{ik}). \tag{1}$$

Explanation for linear model is usually provided as a tuple of learned weights: $(w_0, w_1, ..., w_k)$. If we consider that the features are in the same units (meaning, the data has been normalized beforehand), then magnitude of weights tell us about how important particular features are, while the sign of the weight tells us whether increasing particular feature affects the outcome in a positive or negative way. As an example, visualization of weights for hypothetical loan approval model can be found in figure 2.

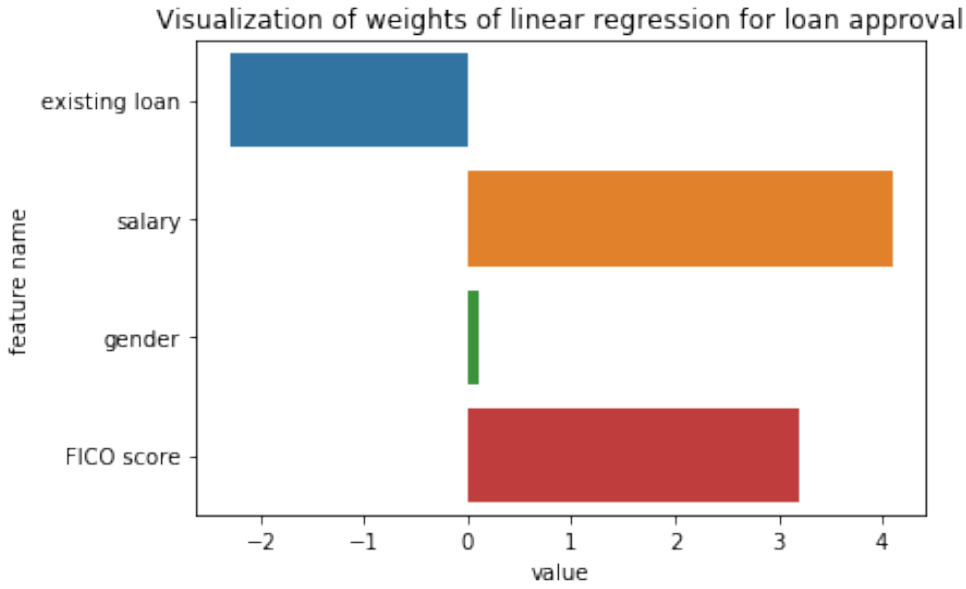


Figure 2: Hypothetical loan approval model. Weights of linear model are provided as an explanation. In this model FICO score and salary have the biggest impact on loan approval, while gender has almost no impact, and existing loan has negative impact.

As we have already mentioned, linear model is globally interpretable only if the number of non-zero parameters stays small (however, how 'small' depends on the application and particular person), which is the case for sparse linear models.

Even though, linear models are inherently globally interpretable, they might not be as expressive and powerful as Deep Learning Models, that is why in a lot of cases Deep Learning Models are preferred. However, there are cases when interpretability is of high importance and that is why linear models are preferred. Interesting example in this category is the model used to generate credit scores. According to FICO report [5], the underlying model for generating FICO credit scores is logistic regression, since it requires high amount of interpretability.

2.3.2 Decision trees

Decision tree has a tree-like structure, each internal node of the tree represents condition and each leaf represents a possible outcome (class outcome). Decision tree represents an algorithm, which only consists of 'if-else' constructions. Exact classification rules are represented by the paths from the root node to the leaves of the tree. An example of decision tree is depicted in figure 3.

Decision trees are considered to be easily interpretable, they also belong to the group of globally interpretable models. People study possible decisions that can be made by following a path from root node to one of the leaves. However, if the decision tree is very deep, then following each path might become a hard task and such a tree ceases to be easily interpretable. So, there is always a trade-off between

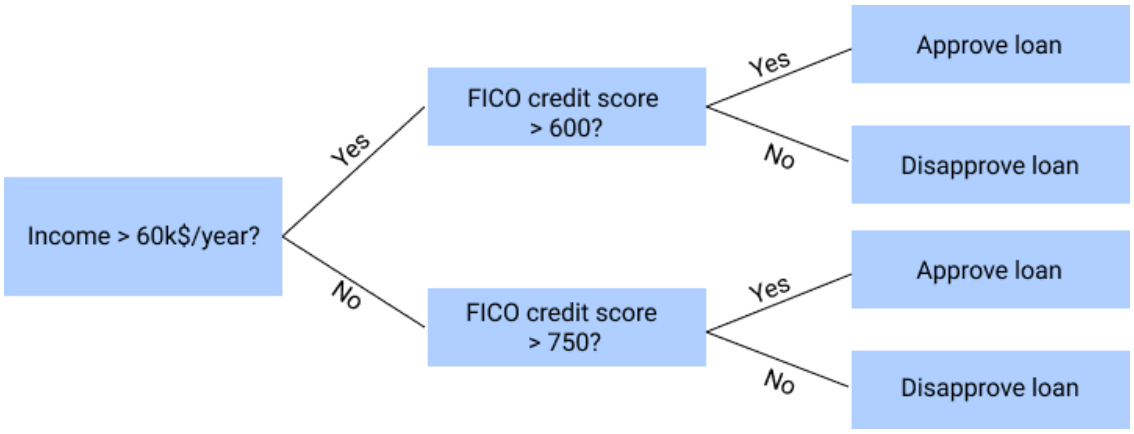


Figure 3: Hypothetical loan approval model represented as a decision tree. In this example, model has 2 classes: 'Approve loan' and 'Disapprove loan'.

how accurately particular tree represents underlying phenomena (usually, deeper decision tree can have better accuracy) and how interpretable this tree is (usually, shorter trees are preferred, when interpretability required).

2.3.3 Decision rules

Decision rule is defined as simply as 'if-then-else' statement, which consists of *condition* and *prediction*. An example of decision rule is 'IF *Income >60k$/year* AND *FICO credit score >600* THEN *Approve loan*, ELSE *Disapprove loan*'. Decision rules can be extracted from decision tree by simply following the path from the root node to one of leaves.

Decision rules, just as decision trees, are considered to be globally interpretable models. However, sometimes decision trees are preferred, because of their clear visual structure.

2.3.4 Decision lists

Decision Lists [29] are used to represent Boolean functions. Suppose that B is a set of Boolean functions on $\{0,1\}^n$. Boolean function f is called a *decision list*, if we can evaluate it in a sequential way. For the given datapoint x, we first evaluate $f_1 \in B$ on this datapoint. If $f_1(x) = 1$, then we return value $c_1 \in \{0,1\}$ (predefined value). If not, we evaluate the next function $f_2 \in B$. If $f_2(x) = 1$, then we return value c_2. We continue this process until some of the functions from the following list $f_1, f_2, ..., f_r$ will return 1. If none of them returned 1, we assign $f(x) = 0$. So, decision list can be thought in terms of 'if-then-else' commands.

```
if f_1(x) = 1:
    f(x) = c_1
else if f_2(x) = 1:
    f(x) = c_2
...
```

```
else if f_r(x) = 1 :
    f(x) = c_r
else:
    f(x) = 0.
```

Decision list is specified as a following sequence $(f_1, c_1), (f_2, c_2), ..., (f_r, c_r)$, such that $f_i \in B$, $c_i \in \{0, 1\}$.

Decision lists are also considered to be globally interpretable models, because of their inherently interpretable structure, just as in the decision trees.

2.3.5 Mimicking behaviour of the black boxes

Not all the machine learning models are *globally interpretable*. For most of the models, including Kernel SVM [10] and Deep Learning [18, 22], there is no way to easily follow the entire reasoning that leads to different outcomes. We call such models *black boxes*.

However, there is still a need to understand reasoning of these more flexible and complex models. One way to tackle this problem is to mimic behaviour of *black boxes* with globally interpretable models such as decision trees or lists.

We can formulate this problem as following. Given a black box classification model b (e.g. Kernel SVM or any Deep Learning Model), the goal is to find globally interpretable model c, which mimics behaviour of the black box b. Such model c serves as a *global explainer*.

For example, if we want to approximate any Deep Learning classification model, we could sample N data points $x_1, x_2, ..., x_N$, evaluate black box model on those data points and obtain labels $y_i = b(x_i)$ for each datapoint x_i. Then we can use this dataset to train globally interpretable model c (e.g. decision tree). This process resembles plain supervised learning approach, however it is important to note that in this case we do not have any restrictions to the input data, because we could sample as many data points as needed and theoretically achieve any accuracy. As usual, in this approach there is a trade-off between how accurately globally interpretable model c mimics black box b and how easy it is to interpret model c. Since, we can sample as big dataset as possible, we could have a good approximation to the black box b, but interpretability might suffer from it.

One of the first works in this field by Craven and Shavlik [11] approximates Neural Networks using decision trees.

Another work by Krishnan et al. [21] also approximates Neural Networks using decision trees, however in this case they have three step method. In the first step with the help of genetic programming, method generates 'prototypes' for each class. In the second step method selects the best 'prototypes', which are utilized for training in the third step. This approach claims to lead to smaller and more understandable decision trees.

Method by Johansson and Niklasson [20] also mimics neural networks with the help of decision trees, but utilizes genetic programming to evolve decision trees. Paper claims to achieve much more accurate trees than by just training model on original training dataset.

Even though, previously mentioned methods were used to explain mostly Neural Networks, they can also be used to explain any other models. Methods, which can be used to explain any machine learning model (e.g. via mimicking it) are called *agnostic*. Methods, which can only be used to explain particular type of model are called *model-specific*. Methods, which are specific to particular type of data are called *data-specific*, for example some methods can only work with tabular data. All the previously mentioned methods for mimicking behaviour of black boxes are *agnostic*.

As an example of *model-specific* method, we could consider method called *SVM + Prototypes* [6]. This method is specific to provide explanations for SVM. It first utilizes clustering algorithm to find prototypes, and then finds ellipsoids in the data space, which are transformed into the decision rules.

2.4 Local explainers

We have already discussed that even in the case of black boxes, it is possible to mimic their behaviour with much simpler globally interpretable models. However, these days very complex models are proposed with millions of parameters, such as Hinton et al. [18], Krizhevsky et al. [22]. For such complex and capable models, it is almost always impossible to mimic their behaviour with such simple models like decision trees with high enough accuracy. That is why different methods are needed in this case.

Local explainers 2.2.1 provide explanations for a given model, which help to establish relationship between particular input components and model's output. For example, if we have a computer vision classification model, and this model for a given cat image outputs class 'cat', then local explainer could provide a modified image, where the most discriminative parts of the 'cat' class are highlighted, as an explanation.

Due to widespread of very complex Deep Learning models, this category of methods has received a considerable attention in recent years [28, 14, 9, 31].

Usually explainers are divided into two categories: *agnostic* and *model-specific*. Agnostic explainers can give explanations to any model, while model-specific methods work with particular type of models (e.g. one can design a method to explain Convolutional Neural Networks (CNNs) [22] by utilizing internal structure of CNNs). Agnostic explainers are able to provide explanations to any model, so they do not utilize internal structure of specific models.

There exists other categories of explainers, which are not fully agnostic, but they are not specific to any particular type of model. These categories of methods usually utilize some property of the black box in order to give a better explanation. Category that recently received considerable attention and is discussed in this thesis: *Gradient-based explainers*. This category of methods rely on the possibility of calculating the gradient of the output of the model. More about these methods in 2.4.2.

2.4.1 LIME

Local Interpretable Model-agnostic Explanations (LIME) [28] is an agnostic local explainer. It provides explanations in the form of interpretable machine learning models. It finds an interpretable model that is locally faithful in the neighborhood of a given datapoint (it means it behaves in the same way in the neighborhood of a given datapoint). LIME workflow is depicted in the figure 4.

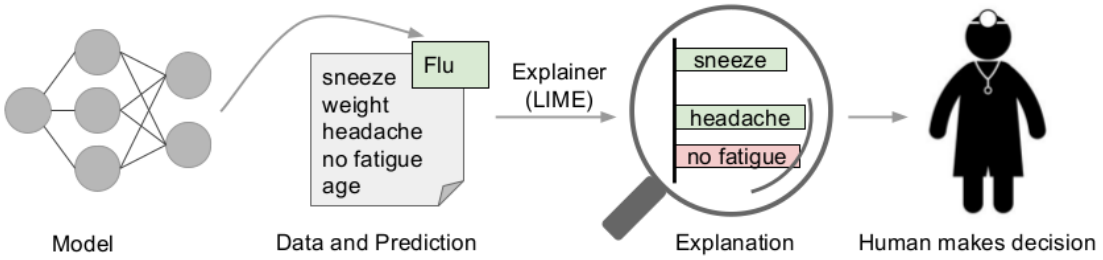


Figure 4: Workflow of the LIME [28] method. In this picture model predicts that patient has a flu and LIME highlights those symptoms, which contributed the most to this prediction.

This paper distinguishes between usual explanations and *interpretable explanations*. The difference is that *interpretable explanation* is a representation that is understandable to humans, regardless of features used for the model. They give an example of *interpretable explanation* for a text classification problem, where as *interpretable explanation* can serve a binary vector, which indicates absence or presence of the word in a sentence, while the machine learning model can use any embedding [25] as an input, which is not comprehensible to humans.

In the same way they introduce *interpretable explanations* for image classification problem as binary vectors, which indicate absence or presence of contiguous set of pixels (superpixel).

In general, they denote input datapoint as $x \in \mathbf{R}^d$, while its interpretable representation as $x' \in \{0,1\}^{d'}$.

Problem specified as follows. Given the model to be explained $f : \mathbf{R}^d -> \mathbf{R}$, set of interpretable models G (e.g. set of sparse linear models), where each $g \in G$ is of the following form $g : \mathbf{R}^{d'} -> \mathbf{R}$, the goal is to find explanation in the following form:

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \mathcal{L}(f, g, \pi_x) + \Omega(g), \tag{2}$$

where

1. $\pi_x(z)$ is a proximity measure between two data points: z and x. It is used to define locality around datapoint x.

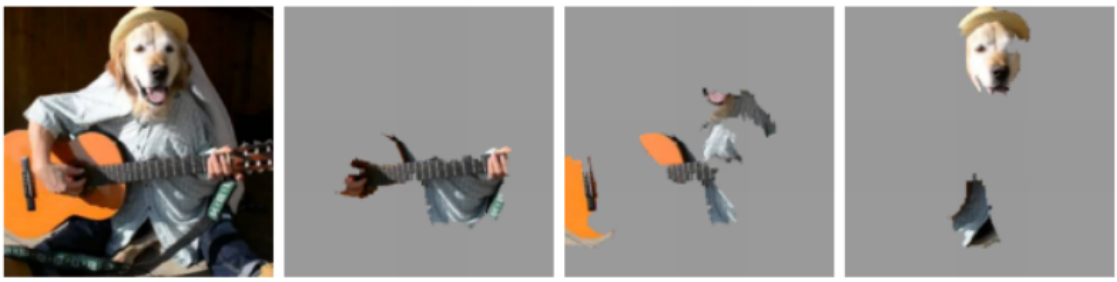
2. $\mathcal{L}(f, g, \pi_x)$ is a measure of how unfaithfully g approximates model f with proximity measure π_x.

3. $\Omega(g)$ is a measure of complexity of model g.

So, they find an explanation as interpretable model $\xi(x)$, which is found as a trade-off between how faithful this model to f around datapoint $x \in \mathbf{R}^d$ and how complex it is.

In a concrete setting for experiments, they use the following functions:

1. $g(z') = w_g \cdot z'$ sparse linear models as *interpretable explanations.*
2. $\pi_x(z) = \exp(-\frac{(x-z)^2}{\sigma^2})$ is an exponential kernel.
3. $\Omega(g) = \infty \cdot \mathbf{1}[||w_g||_0 > K]$ sets a limit K on the number of words to be in explanation.
4. $\mathcal{L}(f, g, \pi_x) = \sum_{z,z' \in Z} \pi_x(z)(f(z) - g(z'))^2$.



(a) Original Image (b) Explaining *Electric guitar* (c) Explaining *Acoustic guitar* (d) Explaining *Labrador*

Figure 5: Explaining a prediction made by Google Inception v3 [37] using LIME [28].

They demonstrate results of the method on Google Inception v3 model [37], results are depicted in figure 5.

2.4.2 Gradient-based explainers

Gradient-based explainers rely on the possibility of calculating the gradient of the output of the model and use it to provide explanations for a given model. This section overviews several gradient-based explainers, including work by Simonyan et al. [33] and Selvaraju et al. [31].

Work by Simonyan et al. [33] introduces gradient-based explainer, which only utilizes knowledge of the gradient, however they show results in the paper only for Convolutional Neural Networks [22]. For a given image x, explanation is given by the saliency map $M \in \mathbf{R}^{m \times n}$. It is computed as follows $M_{ij} = |w_{i,j}|$, where $w = \frac{\partial S_c}{\partial x}$. S_c is the output of the model (probability) for a given class c. The basic idea of the explanation is that it highlights the most class-discriminative parts of the image. Examples of provided explanations by this explainer are depicted in figure 6.

Grad-CAM [31] explainer is model-specific gradient-based explainer, which only provides explanations for Convolutional Neural Networks [22], because it utilizes its structure. They provide explanation in the form of class-discriminative localization map $M \in \mathbf{R}^{u \times v}$, so it has width u and height v. In order to obtain this map, first it is needed to compute coefficients $\alpha_k^c = \sum_i \sum_j \frac{\partial S_c}{\partial A_{ij}^k}$, where S_c is the output of the model

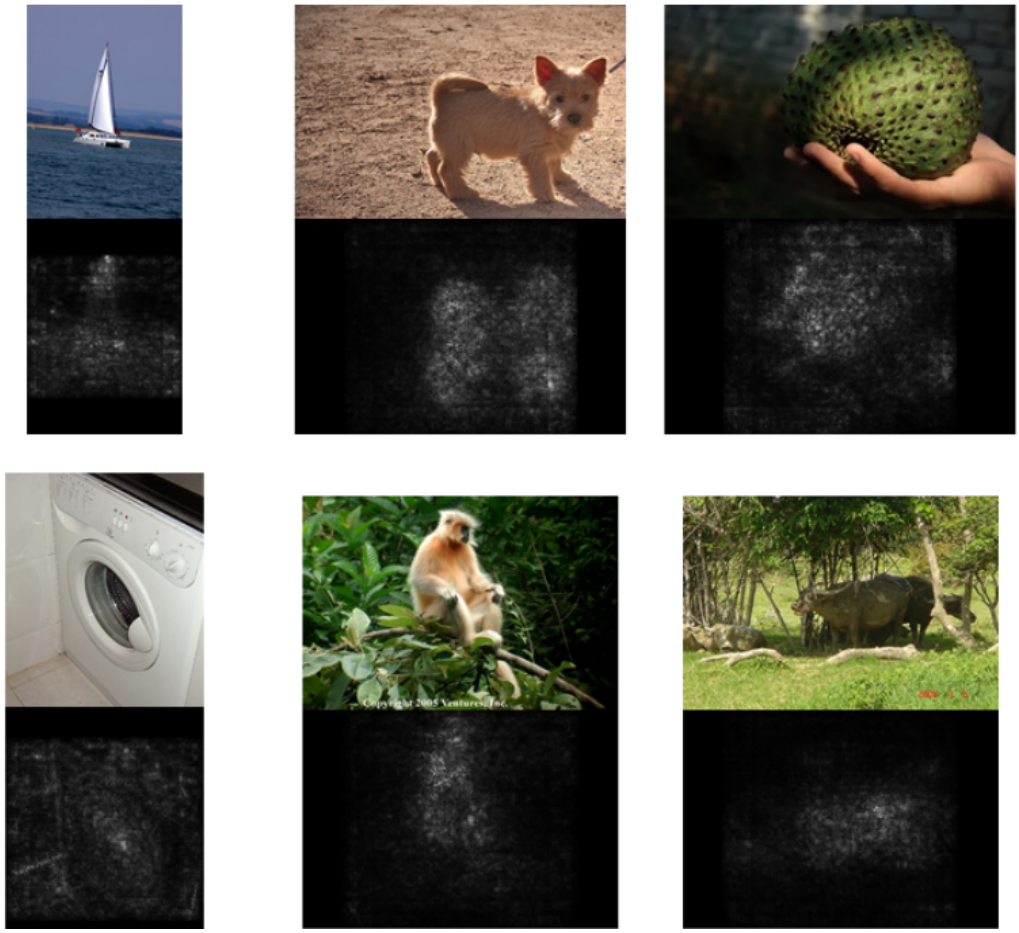


Figure 6: Explanations for several images provided by Simonyan et al. [33].

(probability) for a given class c, A^k - feature map k of a given convolutional layer. Then explanation is provided as follows $M = ReLU(\sum_k \alpha_k^c A^k)$, where $ReLU(x) = max(0, x)$. Examples of provided explanations are depicted in figure 7.

2.4.3 Counterfactual explainers

Miller [26] claims that most of the work in the field of explainable AI is based on the intuition of the researchers of what 'good' explanation is. This paper argues that field of explainable AI should use already existing research in such fields as psychology, biology and cognitive sciences, which study different aspects of providing good explanations. One of the major findings this paper reports is that explanations should be *contrastive*. It means that when some event P happened, people are not interested in why P happened, but rather they are interested in why P happened instead of some other event Q. So, good explanations are sought as a response to particular *counterfactuals*.

Counterfactual Condition is a conditional containing 'if-clause' which is contrary to the fact. Examples:

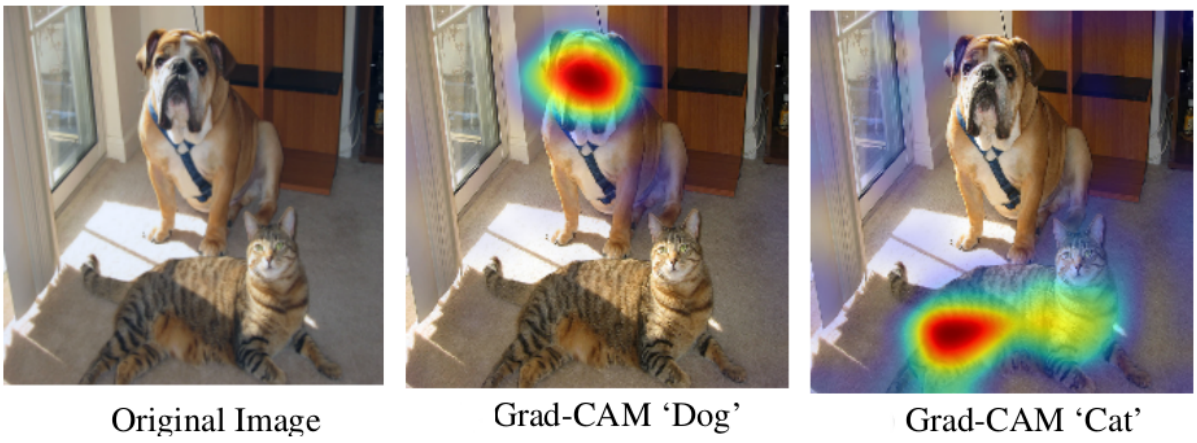


Figure 7: Explanations for several images provided by Selvaraju et al. [31] model-specific gradient-based explainer.

1. If weather *has not been rainy today*, we *would go for a walk*.

2. If I *had not drunk* hot coffee, I *would not have burned* my tongue.

Basically, it is an expression of form: If *CAUSE*, then *EVENT*.
Further, we call *counterfactual* an object, where *CAUSE* is true, s.t. *EVENT* happened. Examples:

1. 'If this cat did not have whiskers, then it would be recognized by classifier as a dog'. Counterfactual in this case is 'cat without whiskers', which consequently is recognized as a dog.

2. 'If I have studied more, I would land this job'. Counterfactual in this case is 'person who studied more', which consequently landed this job.

In machine learning context, we are interested in answering the following question: 'Why this particular object was classified as class A and not class B?'. The *counterfactual* in this case would be a modified object, which classified as class B.
Work by Wachter et al. [38] proposes agnostic gradient-based counterfactual explainer. Given already trained model $f_w(x)$ with trained weights w, point x_0 to which we want to find counterfactual, paper proposes to optimize the following function:

$$\underset{x'}{\operatorname{argmin}} \max_{\lambda} \lambda(f_w(x') - y')^2 + d(x_0, x'), \tag{3}$$

where x' is a counterfactual, y' is desired class for this counterfactual, $d(x_0, x')$ is a distance function, which measures how far the counterfactual x' is from the original datapoint x_0. It is important to note that authors specify target class y' for counterfactual, however it is not always the case, sometimes counterfactuals are acquired by reducing original datapoint highest class probability, without specifying target class for it.

Wachter et al. [38] propose to have distance function $d(x_0, x') = \sum_{k^F} \frac{|x_{0,k} - x'_k|}{MAD_k}$, where MAD_k is median absolute deviation over the feature k. They report results on generating counterfactuals for *PIMA Diabetes dataset* [34]:

```
Person 1: If your 2-Hour serum insulin level was 154.3, you
would have a score of 0.51
Person 2: If your 2-Hour serum insulin level was 169.5, you
would have a score of 0.51
Person 3: If your Plasma glucose concentration was 158.3 and
your 2-Hour serum insulin level was 160.5, you would have a
score of 0.51
```

Fong and Vedaldi [15] propose gradient-based counterfactual explainer for image classification models. This method provides explanations by removing regions, which are the most disriminative for a target class. This method finds mask $m \in [0,1]^\Lambda$ (for each pixel $u \in \Lambda$ there is a scalar mask value $m[u] \in [0,1]$) by solving the following optimization problem:

$$m^* = \underset{m \in [0,1]^\Lambda}{\operatorname{argmin}} \lambda_1 ||1 - m||_1 + \lambda_2 \sum_{u \in \Lambda} ||\nabla m[u]||_\beta^\beta + E_\tau[f_c(\Phi(-\tau x_0, m))], \tag{4}$$

where $\sum_{u \in \Lambda} ||\nabla m[u]||_\beta^\beta$ is the total-variation (TV) norm, and $\Phi(x_0, m)$ (here x_0 is input image and m is a mask) is the perturbation operator. For example, we could define constant perturbation operator as following:

$$[\Phi(x_0, m)](u) = m[u]x_0[u] + (1 - m[u])\mu_0, \tag{5}$$

where μ_0 is the average color. An example of applying this method is depicted in figure 8.

2.5 Model-specific explanations

Work by Zeiler and Fergus [41] focuses on explaining the internal structure of Convolutional Neural Networks (convnets). Paper proposes new visualization technique that provides insights about intermediate convolutional layers of convnets.

Paper considers standard convnet model, which takes 2D image x as an input and outputs probability vector y over C different classes. Architecture of the typical convnet consists of layers, where each layer has:

1. *Convolution operator.* This operator takes as input the output of the previous layer and convolves it with the number of *filters*. Convolution of the output of the previous layer with one filter produces a *feature map*. So, output of the convolution operator is a number of feature maps.

2. *RELU (REctified Linear Unit).* RELU is an activation function, it is defined as $f(x) = max(0, x)$, so it takes result of *convolution operator* and outputs clamps all negative values.

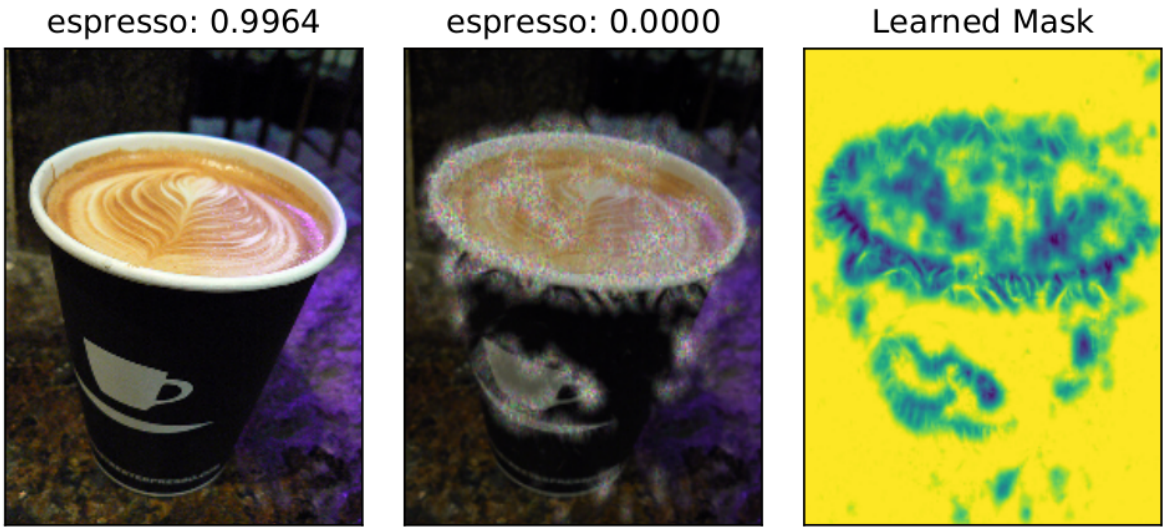


Figure 8: Explanation provided by gradient-based counterfactual explainer [15]. This image on the left is correctly classified using GoogLeNet [36]. Central image is counterfactual which obtained by learning mask (right image) and perturbing image with the noise.

3. *Max pooling [optional].* It is a down-sampling operator, which down-samples representation by taking maximum value over the neighborhood region.

4. *Local contrast normalization [optional].* This operator normalizes the responses among the feature maps.

Typically, the last several layers are fully connected and there is a softmax in the end to ensure that model outputs normalized probabilities. Convnet architecture used by Zeiler and Fergus [41] is depicted in figure 9.

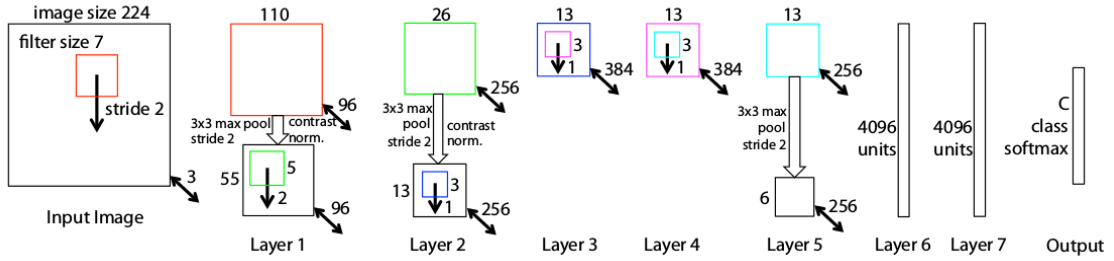


Figure 9: Convnet architecture used by Zeiler and Fergus [41]. This architecture takes 224x224 image and outputs vector of probabilities over C classes. Layers 1-5 are convolutional layers, while layers 6-7 are fully connected. Details about those layers are discussed in section 2.5.

To examined trained convnet, they propose to compute output of each layer for the given image. Then, in order to obtain an insight for a particular feature map in

a given layer, they set all other activations (outputs of the layer) to zero. After this, they recover an image in the image space from the activations in a given layer using deconvolutional network (deconvnet) [40]. The idea of deconvnet is to revert back the process of a given convnet (meaning, pass output of the layer (number of feature maps) as input and output a restored image in the input image space), it consists of *unpooling*, *rectification* and *filter reconstruction* operations:

1. However, *max pooling* is non-invertible, but we can have an approximation for the inverse by saving the location of the maximum within each region of pooling. This set of maximums is usually referred to as set of switch variables. Process of approximating max pooling operator is depicted in the bottom of figure 10 and is called *unpooling*.

2. *Rectification* is the same as applying RELU operator to ensure non-negative features.

3. *Filter reconstruction* is using transposed (flipped vertically and horizontally) versions of the same filters and applies them to the rectified feature maps. This operation sometimes is also called *fractionally-strided convolution*.

Illustration of the convnet architecture and *unpooling* operator is depicted in figure 10.

Results of using proposed technique is depicted in figure 11. In this figure, for each feature map in a given layer, top 9 activations are visualized, which are projected to the image space using deconvnet. This visualization reveals that each feature map is responsible for particular *concept*, which can be visualized (lower layers learn edges/corners and different shapes, while deeper layers capture texture and high-level concepts like dog face, bird's leg, etc). Additionally, they also present image patches that caused those activations.

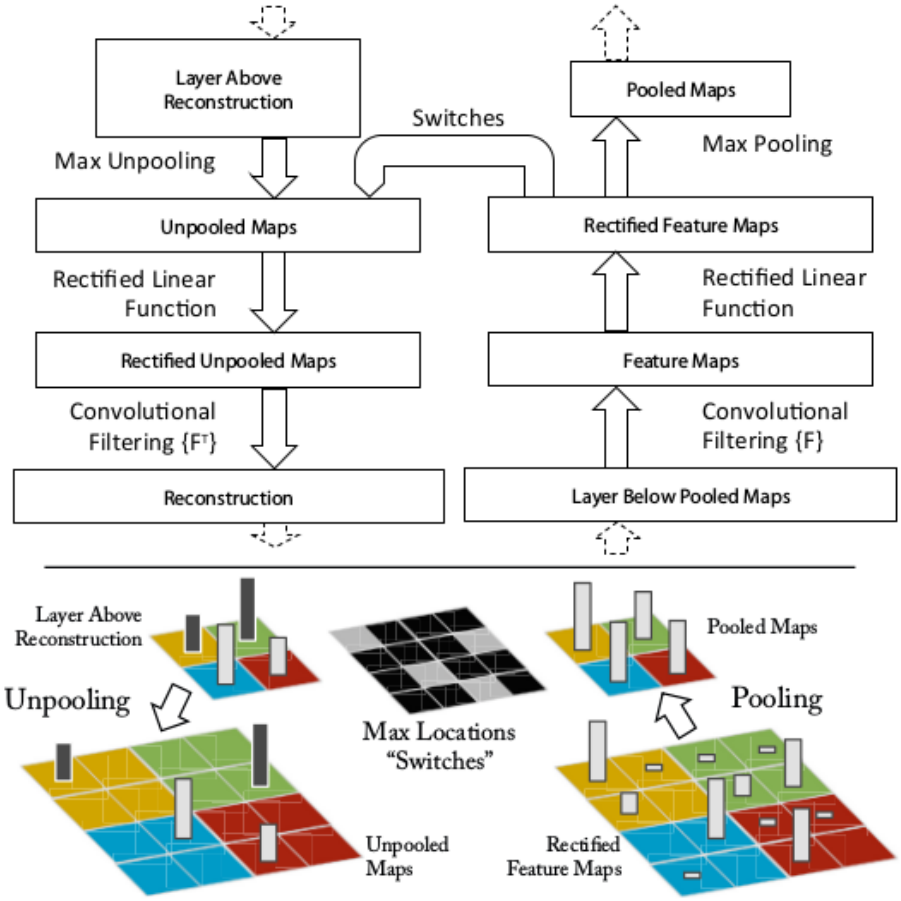


Figure 10: On top is an illustration of convnet layer (right) and deconvnet layer (left). On bottom is an illustration of *unpooling* operation described in section 2.5. Figure from the work by Zeiler and Fergus [41].

Figure 11: Visualization of feature maps using technique proposed by Zeiler and Fergus [41] and discussed in section 2.5. For each layer, feature maps are visualized by their top 9 activations, which are projected to the image space using deconvnet. Also, image patches that cased those high activations are presented.

3 Framework

In this section we propose a new framework for explaining machine learning classifiers. Previously mentioned local explainers [15, 38, 31, 33, 28] aim to explain classifier's decision only for a particular given object. For example, in the case of image classifiers these methods are aiming at explaining classifiers by highlighting parts of the given image, which are the most discriminative for specific class as in figure 8. So, these methods do not provide a *global* perspective for a given class.

Work by Ribeiro et al. [28] argues that providing an explanation for a single instance does not give a *global* understanding for a given model. This work proposes method *SP-LIME*, which is based on submodular optimization, it samples a set of representative instances. Then, explanations are manually evaluated for each sampled instance.

In this thesis, we are taking a different perspective on obtaining a *global* perspective for a given class. In our approach, for a given set of data points we are finding set of counterfactuals, which are obtained by almost the same semantically meaningful perturbations to given set of data points. Semantically meaningful perturbation means that the mask used to do this perturbation has semantically meaningful entries. In case of images, this concept illustrated in figure 12. In this example, top row are input images, and bottom row is the result of applying given mask. Note that one entry of the mask (e.g. eyes) could correspond to different locations in different images (it also might correspond to nothing, if this unit is not presented in the image).

Now we formally define the proposed framework. Suppose we have a set of input data points $x_1, x_2, ..., x_n$ for which we want to find set of counterfactuals. We do this by solving the following optimization problem:

$$\underset{m_1,m_2,...,m_n}{\operatorname{argmin}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} f_{target}(\Phi(x_i, m_i))}_{term_1} + \lambda \underbrace{\frac{1}{n}\sum_{i=1}^{n} norm(m_i)}_{term_2} + \mu \underbrace{\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} d(m_i, m_j)}_{term_3}, \quad (6)$$

where

1. m_i is a learned mask for input datapoint x_i. Each datapoint has its own mask.

2. $f_{target}(x)$ is the output of a given model for a given *target* class. For example, in the case of neural network it could be neuron in the last layer corresponding to particular class.

3. $\Phi(x, m)$ is a perturbation operator applied to object x using mask m. The output of perturbation operator is a perturbed object x, which we call *counterfactual*.

4. $norm(m)$ is a norm of a given mask m. In out experiments we are using L_1 norm to achieve sparsity, if not specified otherwise.

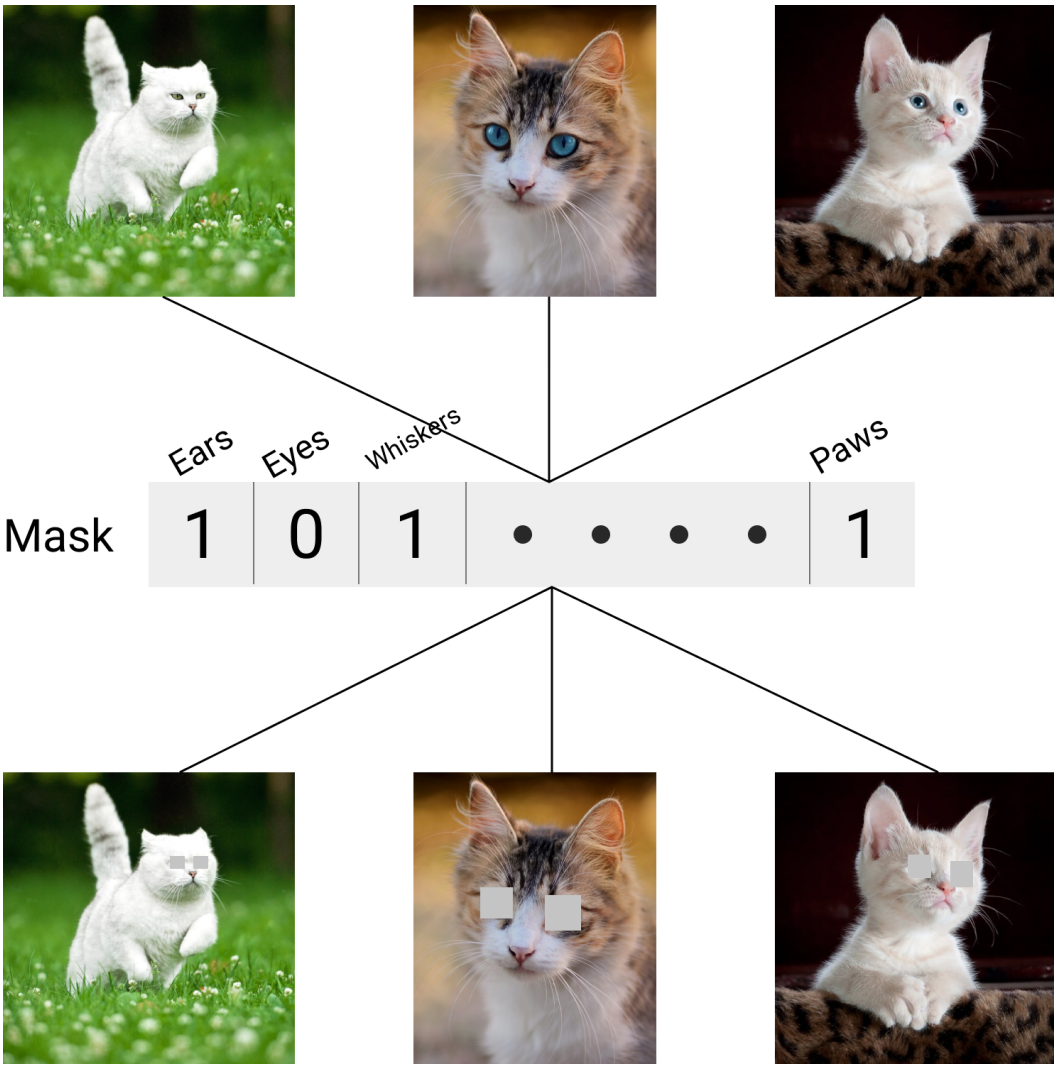


Figure 12: Example of applying a meaningful perturbation to each image in the top row. Entries of the mask are meaningful units of the image. In this case entry '1' means that this unit is presented in an image, while '0' is the opposite. Note that one entry of the mask could correspond to different locations in different images (it also might correspond to nothing, if this unit is not presented in the image). Note that perturbation operator could be also different. In this case constant perturbation operator applied as in equation 5.

5. $d(m_1, m_2)$ is a distance between two masks m_1 and m_2. If masks are semantically meaningful as in figure 12, then in out experiments we define distance between masks as a simple L_2 distance. However, in the case, when masks are not meaningful, distance could be defined in a more complex way to reflect that both masks should correspond to the same meaningful part of the object.

6. λ and μ are hyperparameters that control sparsity of the masks and their proximity to each other respectively.

We provide an explanation as $m^* = \frac{1}{n}\sum_{i=1}^{n} m_i$, which is the average of all masks. Note that bigger μ leads to less variation in m_i.

In the provided optimization problem 6, $term_1$ is responsible for reducing target class probability of found counterfactuals by optimizing the masks, $term_2$ is responsible for keeping norm of masks small (in out experiments we use L_1 norm to ensure sparsity), $term_3$ is to make sure that masks do not deviate too much from each other, because our goal is to obtain a *class explanation* and not an explanation for every particular instance.

In order to apply proposed framework to a given classification model to explain given target class, we need to specify:

1. Form of the masks m_i.
2. Perturbation operator $\Phi(x, m)$.
3. Norm of the mask $norm(m)$.
4. Distance between two masks $d(m_1, m_2)$.

In the following subsections, we propose several ways of specifying these for *tabular data*, *image data* and *text data*, which will be used later in the experiments section.

Input data points $x_1, x_2, ..., x_n$ and hyperparameters λ, μ are specified and tuned individually for a particular dataset/problem. For every experiment we show which data points and hyperparameters we chose.

3.1 Tabular data

In the simple case, when we have interpretable features and can modify any feature with any value, we could specify components of the framework as following:

1. Each mask m_i has the same dimensionality as x_i, so $dim(m_i) = dim(x_i)$.
2. Distance between two masks is L_2, so $d(m_1, m_2) = ||m_1 - m_2||_2$.
3. Norm of the mask is L_1 norm, so $norm(m) = ||m||_1$.
4. Perturbation operator is simply $\Phi(x, m) = x + m$.

Note that in this setting we allow *any* modifications of features to any datapoint, so this setting provides us with the biggest amount of flexibility in terms of applied perturbations. Mask in this case acts as a vector of changes that are needed to be applied in order to get a counterfactual.

Even though this setting is very simple and useful, and it provides interesting insights about the model, it might not always be suitable for several reasons:

1. It might be not desirable to apply changes to particular features. An example could be a banking loan approval model, which takes customer data (income, credit score, age, gender, etc) and outputs the decision (approve/disapprove

loan). If the customer got disapproved with his loan application, we might use the proposed method to provide an explanation to the customer why his loan application got rejected (e.g. according to GDPR 1.3). In this case there is a set of protected features, which we can not change, including *age* and *gender*.

However, sometimes it might be even desirable to have changes to such protected features. For example, in the case of validating model and identifying biases, such as those in 1.1, studying affection of model's outcome by such protected features might be highly desirable. In this case, if our model changes prediction from *Disapprove loan* to *Approve loan* just by changing the *gender* feature, then out model most probably discriminating against *gender* and this case should be handled appropriately.

2. Sometimes it only makes sense to perturb input data points by changing several features simultaneously, otherwise it could lead to very improbable counterfactuals. An example of such application could be classification model over apartment objects, which takes as an input characteristics of the apartment (e.g. number of bedrooms, number of restrooms, apartment size in square meters, etc). In this case modifying number of bedrooms of the apartment should most probably affect apartment size, so it might make more sense to change those simultaneously.

One way to tackle both of the previously mentioned reasons is to introduce m *individual perturbation functions*: t_1, t_2, ..., t_m. Each perturbation function $t_i(x, \theta_i)$ takes input datapoint x and its argument θ_i, and outputs corresponding perturbation. Then we define perturbation operator as $\Phi(x, m) = t_1(t_2(...t_m(x, \theta_m), ..., \theta_2), \theta_1)$. In this case mask will be represented as $m = (\theta_1, \theta_2, ..., \theta_m)$ and might have different dimensionality from the input data points.

Note that if we set $m = n$ and

$$t_i(x, \theta_i)[j] = \begin{cases} x^j + \theta_i, \text{if } i = j \\ x^j, \text{if } i \neq j, \end{cases}$$

where x^j is the j-th feature of x. Then this approach is equivalent to the simple case mentioned in the beginning of this subsection 4.1, because t_i only modifies i-th feature.

Using this approach, it is straightforward to prevent some features from being changed, e.g. if we want to prevent feature j from being changed, we set $t_j(x, \theta_j) = x$. It is also straightforward to define t_i in a way that changes several features simultaneously. However, note that in this case order of applying *individual perturbation functions* might affect the output of $\Phi(x, m)$.

3.2 Image data

In order to apply proposed framework to explain classes of image classification models, we need to have semantically meaningful entries of the mask as in figure 12. The problem with this approach is that we need to have a mapping from entry of the

mask to the part of the image with this entry (meaning, if we have an entry of mask corresponding to the 'eyes', then we need to know for every image where eyes are located). That is why there is no straightforward way to apply already existing methods [33, 31, 15], since their masks are based on location (e.g. if we apply one of mentioned methods to find counterfactuals for two different images and it visually found 'eyes' in both images, there is no straightforward way to quantify the distance between two masks, since location of eyes in the image could be different).

To tackle this problem, one might just manually label different parts of every image or use existing dataset such as COCO [24], which has 91 different categories of objects already labeled for every image. However, this has an obvious drawback of doing manual work with labeling our images, or in the case if we are using COCO – it restricts us to have explanations only to those images that are in the COCO dataset, and have explanations in terms of predefined categories.

Another way to tackle this problem is to utilize internal structure of the model being explained. In this thesis, we utilize internal structure of Convolutional Neural Networks (convnets), since it has received a wide usage in different applications [23, 39, 42].

Several works [41, 33, 13] showed that deeper convolutional layer capture high-level *concepts*, which we discussed in section 2.5. So, every feature map of the convnet is responsible for a particular *concept* (see figure 11, however each feature map retains spatial information. Each feature map is a matrix of neurons, where each neuron is responsible for a *concept* in a given location. We can get rid of that spatial information by aggregating those values (taking *sum* or *mean* operator) for each feature map, which leaves us with an array of values, where each value represents how much particular *concept* is represented in the given image.

Now we define it formally. For the given input images $I_1, I_2, ..., I_n$, we preprocess them by passing through all convolutional layers of our model and we obtain stack of feature maps for every image. Preprocessing step defines a mapping $conv : I \to \{A^k\}_{k=1}^P$, where A^k is the k-th feature map in the last convolutional layer, P is the number of feature maps in the last layer. We obtain our input x_i by applying preprocessing step $x_i = conv(I_i)$, and $x_i = (x_i^1, x_i^2, ..., x_i^P)$, where x_i^j is the j-th feature map for i-th datapoint. We define masks as $m_i \in \mathbf{R}^P$, so it has the same dimensionality as the number of feature maps P. We define perturbation operator $(\Phi(x, m))[i] = x^i + m$. Note that x^i is a feature map (matrix) and m_i is a scalar, so operation $x^i + m_i$ adds constant value m_i to each element of x^i.

For the given specification, each entry of the mask corresponds to one feature map, which can be interpreted by the method proposed by Zeiler and Fergus [41] and discussed in section 2.5. That is why we consider that mask has semantically meaningful entries and can be used in our framework to provide explanations for image classifiers.

3.3 Text data

Case of text data is very similar to the case of *tabular data* 4.1. We can represent each document as a bag of words. We first build vocabulary V of size K, then we

define $x_i \in \mathbf{N}_+^K$ to be a bag of words representation for i-th document. We define $x_i[j]$ as number of times word with index j appears in document i. Then we can define mask to have the same dimensionality as x_i, so $dim(x_i) = dim(m_i) = K$, and perturbation operator as $\Phi(x, m) = RELU(x + m)$. We use $RELU$ operator here, because counterfactual for x should have non-negative entries.

In this setting, negative value of the mask entry represents the word, which is discriminative for the target class. Since size of vocabulary K could be very big (millions of words), we propose to only look at top-N smallest entries of the mask, which will provide top-N words, which are the most discriminative for the target class. In conducted experiments in section 4.3 we use top-7 and top-10 words to provide explanations for instances and classes.

4 Experiments

We conduct and show different experiments for *tabular data*, *image data* and *text data* for proposed framework. Every subsection inside those sections corresponds to a different type of experiment. Code with reproducible experiments can be found at GitHub Repository [3]. We used PyTorch [27] as the main framework for our experiments.

In *tabular data* section 4.1 we analyze role of hyperparameters λ and μ of the proposed framework (see section 3 for details) in 4.1.1, analyze sensitivity of hyperparameter λ in 4.1.2 and show robustness of the proposed framework with respect to the number of input data points in 4.1.3.

In *image data* section 3.2 we visualize explanations for classes with aggregated maps in 4.2.1 and comparing visualizations for different models of VGGNet family in 4.2.2.

In *text data* section we display top words for target classes in 4.3.1 and show consistency of explanations in 4.3.2.

4.1 Tabular data

For our experiments in this section we use *Pima Indians Diabetes* dataset [4]. Dataset consists of 768 data points and 9 features.

We split the data into train/val/test with the ration 0.64/0.16/0.2 respectively. We normalize the data by subtracting mean and dividing by standard deviation.

As a model we train 3-layer neural network with 20 neurons in the first 2 layers and 1 unit in output layer. We use RELU activation for hidden layers and sigmoid activation for the output layer. We use binary cross-entropy as a loss function. We train the model using gradient descent with learning rate 0.03 and weight decay 0.01. Achieved accuracy on the test set is $\sim 78\%$.

4.1.1 Role of hyperparameters λ and μ

In this experiment we analyze how hyperparameters λ and μ affect learned masks. Both of them are defined in equation 6. λ is responsible for magnitude and sparsity of the masks, while μ for proximity of the masks to each other.

To analyze their role, we visualize the masks for 3 cases:

1. $\lambda = 0, \mu = 0$
2. $\lambda > 0, \mu = 0$
3. $\lambda > 0, \mu > 0$

We learn the masks by solving optimization problem proposed in the section 4.1 using stochastic gradient descent with learning rate 0.1. We choose 10 data points from test dataset, which have the biggest probabilities for target_class $= 1$. Visualization of the learned masks for these 3 cases is depicted in figure 14. In this

figure, each row represents an instance and each column represents a feature of the learned mask.

We can see that when both $\lambda = 0$ and $\mu = 0$, there is no clear distinction which features are the most discriminative for our target class, even though we see 'purple' areas for feature 1 and 5. However, there is still a lot of noise. This case is equivalent, when we find these 10 counterfactuals independently, that is why observing counterfactuals 1-by-1 might be misleading, since different counterfactual will indicate different 'important' features (it is easy to notice that for some data points some features might be very important, while for others not at all). This is the typical results that we get with the most of counterfactual explainers, since they are only *local* explanations and not *global*. The usual guidance is to sample many different counterfactual and understand what is in common between them.

For the case when $\lambda = 0.05$ and $\mu = 0$, it does not change the situation much from the previous one, there is still a lot of noise.

When we set $\lambda = 0.03$ and $\mu = 0.2$, we immediately see the structure. Setting μ to non-zero value, forces counterfactual masks to be very similar to each other, that is why they are exploring the common patterns that are relevant to target class in *all* data points and ignoring the noise from *individual* data points. Our intuition here is that setting hyperparameters to non-zero values and having big enough (10-15) amount of data points provides robustness of the masks and can serve as a good explanation for a given target class. We confirm this intuition with experiment in 4.1.3.

In order to see difference between learned masks, we visualize data points from the test set using features 1 and 5 (the ones that have the biggest impact on target class (class 1)) in figure 13. We visualize learned masks as arrows. Cyan arrows represent masks which we learned using $\lambda = 0, \mu = 0$ and we can see that they vary among the data points and there is no single direction. Magenta arrows represent masks which we learned using $\lambda = 0.03, \mu = 0.2$ and we can see that there is very clear consistent direction among all the data points.

4.1.2 λ sensitivity

In this experiment, we analyze sensitivity of hyperparameter λ, while keeping all other parameters constant. We use exactly the same setting as in the previous experiment. Here our goal is to understand how magnitude of λ affects the learned masks.

Visualization of the learned masks for several values of lambda $(0.2, 0.1, 0.05, 0.02)$ is depicted in figure 15. We set $\mu = 0.2$ in every case.

From this figure, setting smaller value of λ requires more features of the mask to have non-zero values (meaning, makes those features discriminative for a target class), however it also reduces mean probability of the target class. Illustration of this concept is depicted in figure 16. When all masks are set to 0, mean probability is $\sim 94\%$, but we can reduce it down to 41.2% by just changing 1 feature and down to 2.8% by changing just 4 features (with very small changes to couple of others).

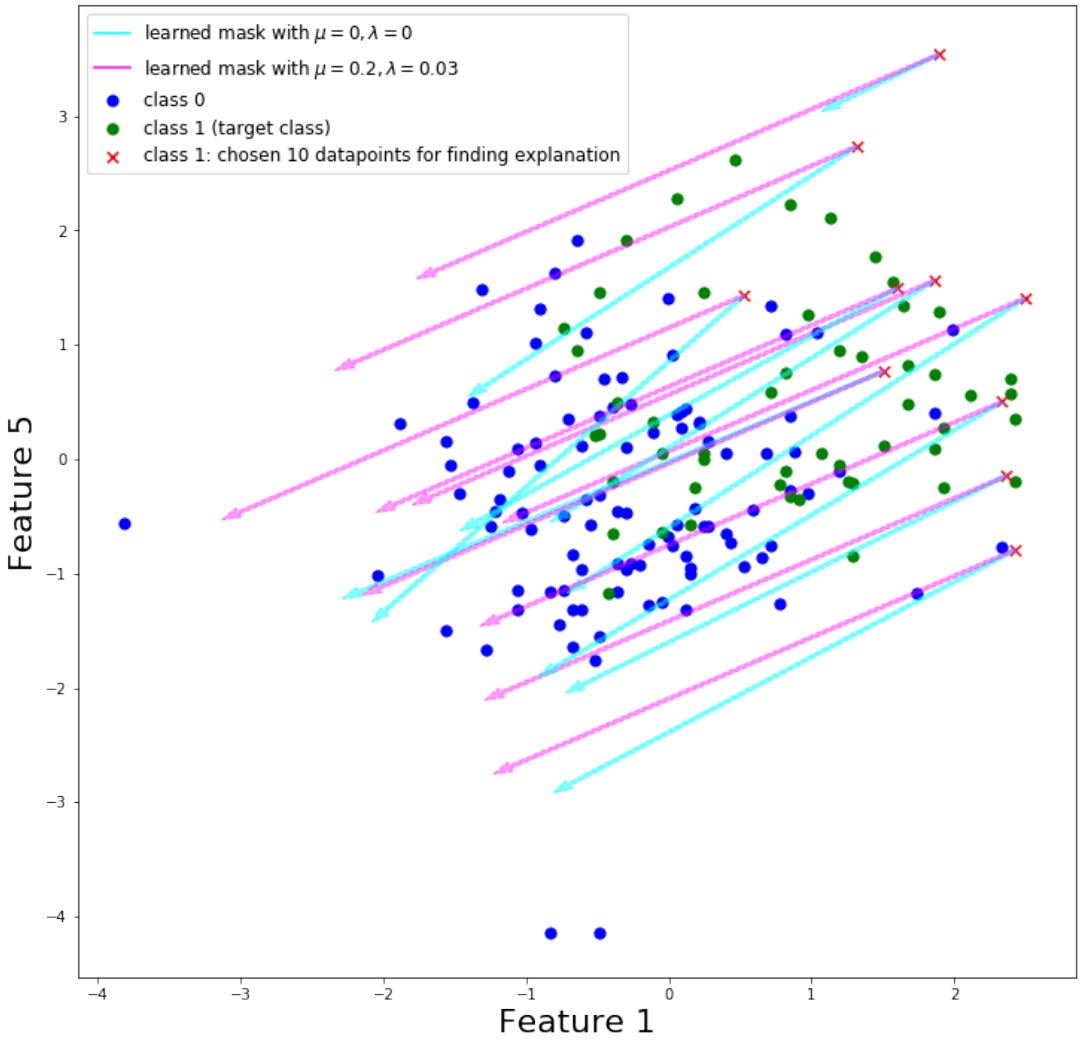


Figure 13: Visualization of data points from the test dataset along with the learned masks using 2 different settings: $\lambda = 0$ $\mu = 0$ and $\lambda = 0.03$ $\mu = 0.2$. Cyan arrows represent masks which we learned using $\lambda = 0, \mu = 0$, while magenta arrows represent masks which we learned using $\lambda = 0.03, \mu = 0.2$. We can see that for magenta arrows there is very clear consistent direction among all the data points, which is not the case for cyan arrows.

4.1.3 Robustness with respect to number of data points

In this experiment we analyze the robustness of our framework with respect to the number of data points. We fix $\lambda = 0.02$ and $\mu = 0.2$, but keep number of input data points different, n from equation 6. We train masks for several values of n: $1, 2, 5, 10, 15, 20, 30$ using the same setting (model and training process) as in the previous experiments.

Visualization of the trained masks for different number of data points is depicted in figure 17. We can see that starting from $N = 10$, the masks start to generalize and new data points do not affect their values, which shows robustness of the framework

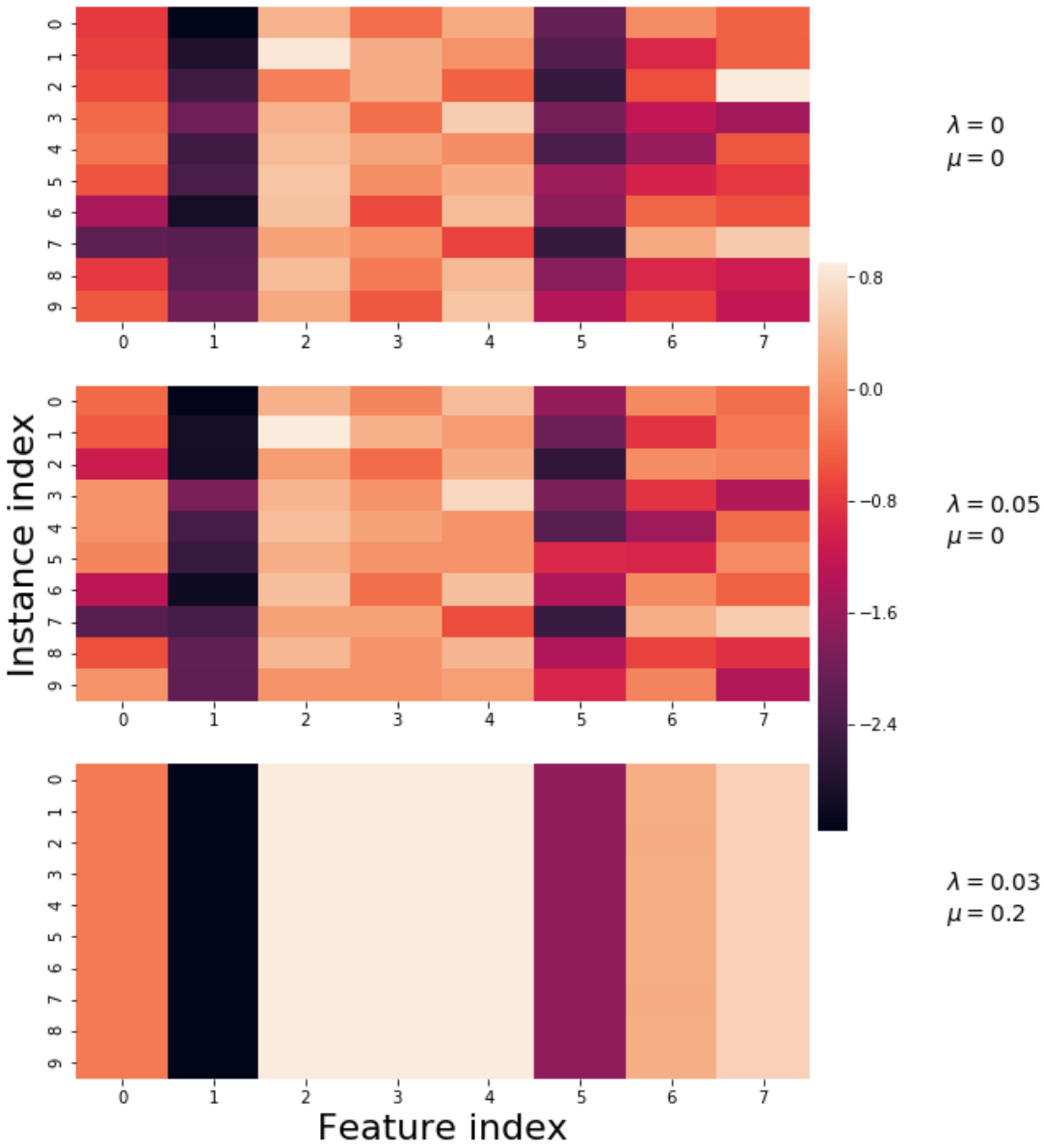


Figure 14: Visualization of learned masks for 3 cases: $\lambda = 0$ $\mu = 0$ (top), $\lambda = 0.05$ $\mu = 0$ (middle), $\lambda = 0.03$ $\mu = 0.2$ (bottom). Each row represents an instance (we have 10 instances in every case) and each column represents a feature of the learned mask. Setting both λ and μ to non-zero values (bottom) helps us to achieve consistency across all 10 masks and provides robust explanation for a target class.

with respect to number of data points for a sufficient enough number of data points. We also sampled new 30 data points and learned new masks for them, but they were pretty much the same as the ones shown in the figure, which shows robustness of the framework and because of that, in our opinion, serves as a good explanation for the target class.

4.2 Image data

For our experiments we use classifiers trained on the subset of *ImageNet* dataset [12], which has approximately 1 million labeled images with 1000 categories for ILSVRC challenge [30].

In our experiments we use different configurations of VGGNet [32]: VGG11, VGG13, VGG16, which are depicted in figure 18.

For learning masks, we used gradient descent with learning rate 0.1 and the same hyperparameters for all experiments: $\lambda = 0.05$ and $\mu = 0.2$.

4.2.1 Visualizing explanations for classes

In this experiment we visualize the learned masks for several classes for VGG16 neural network (architecture is depicted in figure 18). As described in section 3.2, dimensionality of every learned mask equals to the number of feature maps in the last layer. In the case of VGG16, there are 512 feature maps in the last layer. Since number of them is not very small, as in tabular data experiments 4.1, one way to visualize those maps is to aggregate them. As an aggregation method, we consider linear combination of mask entries with feature maps themselves:

$$AM_i = RELU(\sum_{j=1}^{P} m_i^k A_i^k), \tag{7}$$

where m_k^i is k-th entry for i-th learned mask (meaning for i-th image), A_i^k is the k-th feature map for the i-th image, AM_i is an aggregated feature maps for i-th image. $RELU$ here is used to suppress negative results. Note that AM_i is a 7×7 image, so we upsample it to match the input of VGG16 network, which is 224×224. Similar technique was used in Grad-CAM paper [31] and showed good results.

We visualize every AM_i as a heatmap for every image in the dataset. Visualizations are provided for 3 classes: 'Tiger cat', 'Lynx' and 'Arctic fox'. For every class we visualize 7 random images. Results are depicted in figure 19. Every heatmap represents the most discriminative region for a given class. For example, for a class 'Arctic Fox' VGG16 primarily looks at the face of the fox. Having such visualization can provide us an idea about which parts of the image the network looks at, when classifying this object belonging to particular class.

We also made visualization for image, which contains more than 1 object and visualized it. Result of visualization is depicted in figure 20. We can see from this visualization that VGG16 mostly makes its decision by observing the cat face, when describing class 'Cat', however there is a little weight on the dog's neck. When explaining class 'Dog', most of the weight is on the dog's face, but there is also little weight on the cat's face.

Looking at the distribution of weights on the image can also helps in troubleshooting, when classifier made a wrong prediction. Example is depicted in figure 21, where VGG16 gives the biggest score to wrong 'Egyptian cat' class. After observing the heatmap it is clear that it look at the bottom of the image and texture there actually resembles texture of Egyptian cat.

4.2.2 Comparing visualizations for different models

In this experiment we compare different configuration of VGGNet: VGG11, VGG13, VGG16, which are depicted in figure 18. We use the same visualization method as in subsection 4.2.1. We compare provided visualizations for different classifiers for 2 classes: 'Tiger cat' and 'Arctic fox'. Results for class 'Tiger cat' are depicted in figure 22, and for class 'Arctic fox' are in figure 23. Nevertheless, there are big overlaps in terms of explanations for different classifiers, there are still clear differences. For example, for class 'Tiger cat' on rows 3 and 4 VGG16 does not take into account cat's face, while VGG11 and VGG13 do. Another example for class 'Arctic fox', where on row 5 VGG11 almost does not take into account left fox, VGG13 takes into account a lot of snow, while VGG16 only looks at 2 foxes.

4.3 Text data

In this section we use *20 NewsGroups* dataset [1] for our experiments. Dataset consists of $\sim$ 19000 documents and 20 different categories.

As a model we train 3-layer neural network, where 2 hidden layers have 100 neurons each and output layer has 3 neurons. Output layer has 3 neurons, because we trained network on a subset of data points with 3 classes: 'comp.graphics', 'sci.space', 'rec.sport.baseball'. We build vocabulary from training dataset without any data preprocessing, except lower case every word. Size of vocabulary: 119930. We use bag-of-words as an input to the model. Activation function for hidden layers is RELU, for output layer – softmax.

4.3.1 Displaying top words for classes

In this experiment we display the most discriminative words for every class. We learn the masks using proposed technique with $\lambda = 0.05, \mu = 1$ for every target class. As described in section 4.3, dimensionality of every mask equals to the dimensionality of vocabulary, which is 119930 is out case. Because of such huge dimensionality, we only display the top words for each class. We consider top words to be those which have the smallest mask entries, because their deletion from documents will lead to the biggest probability drops of the target class.

In table 1 we display top 10 words for each class. We can see that class 'comp.graphics' is pretty well described with computer graphics related words, but class 'sci.space' definitely has some undesirable words, e.g. 'sci\nlines:'. It seems that network tried to capture only 'sci', but since we have not done any preprocessing, it also captured the last part '\nlines:' of it, because they usually go together in documents. In order to confirm this reasoning, we display such particular document from the dataset in figure 24, which contains 'distribution: sci\nlines 10'. Also, note that email address 'prb@access.digex.com' appears as the top word, which is probably not desirable either. This email address is among top words, because it appears frequently among 'sci.space' documents (it also appears in figure 24). So, explaining classes with the top words using proposed framework might be also helpful in troubleshooting.

When the new model is trained, we might describe every class with proposed framework and make sure that all the top words in every class make sense for this particular class. It helps us to ensure trust in the model, which is discussed in section 1.4, since high accuracy on the test set does not guarantee that model learned anything useful, because both training and test datasets might be biased, but it might be hard to notice due to the size of the dataset and size of the documents. That is why it is always helpful to look at other methods, which help to understand the underlying model better and ensure trust.

Target class	**Top-10 words**
comp.graphics	'graphics', 'video', 'algorithm', 'vga', 'rgb', 'image', 'output', '256', 'package', 'rumours'
rec.sport.baseball	'baseball', 'jewish', 'phillies', 'sox', 'stats', 'pitcher', 'fan', 'ryan', 'win', 'yankee'
sci.space	'planets', 'orbit', 'observatory', 'sci\nlines:', 'prb@access.digex.com', 'temporary', 'moon', 'nasa', 'orbit,', '(pat)\nsubject:'

Table 1: Top 10 words for each target class. Top words are those which have the smallest mask entries.

4.3.2 Consistency of explanations

In this experiment we show consistency of provided explanations (learned masks) when using both $\lambda > 0$ and $\mu > 0$. We randomly choose 10 data points from the test dataset from the category 'comp.graphics' and learn two sets of masks for them: using $\lambda = \mu = 0$ and $\lambda = 0.05, \mu = 5$ (case $\lambda = \mu = 0$ means that we learn masks independently for each datapoint). Then, we randomly choose 5 data points out of 10 and provide explanations using top-7 words for each mask in the same way we did in section 4.3.1, except that in this experiment we report top words *per instance*, not *per class*.

Results of experiment are reported in table 2. It is easy to see the consistency among the top words for $\lambda = 0.05, \mu = 5$ case. For the case $\lambda = \mu = 0$, we can see there are some common words among explanations, e.g. 'computer', 'package', 'file', but there are some which are specific to particular text, e.g. '|\n', 'address', and which are not specific to 'comp.graphics' topic.

This experiment is similar to the one conducted in section 4.1.1 (see figure 13). Basically, proposed framework helps to capture consistent explanations among many data points and takes away the need to manually inspect each datapoint to get *global* understanding of what model learned about the target class.

Text	**Top 7 words** ($\lambda = \mu = 0$)	**Top 7 words** ($\lambda = 0.05, \mu = 5$)
Text 1	'computer', 'package', 'file', 'card', 'graphics', 'pc', 'polygon'	'computer', 'package', 'file', 'card', 'pc', 'algorithm', 'polygon'
Text 2	'computer', 'package', 'card', 'file', 'pc', 'algorithm', 'graphics'	'computer', 'package', 'file', 'card', 'pc', 'algorithm', 'polygon'
Text 3	'computer', 'package', 'card', 'file', 'pc', '|\n', 'packages'	'computer', 'package', 'file', 'card', 'pc', 'algorithm', 'polygon'
Text 4	'computer', 'package', 'file', 'pc', 'algorithm', 'card', 'polygon'	'computer', 'package', 'file', 'card', 'pc', 'algorithm', 'polygon'
Text 5	'computer', 'algorithm', 'pc', 'package', 'address.', 'file', 'card'	'computer', 'package', 'file', 'card', 'pc', 'algorithm', 'polygon'

Table 2: Top 7 words reported *per instance* in 2 settings: $\lambda = \mu = 0$ and $\lambda = 0.05, \mu = 5$. Top words are those which have the smallest values of mask entries.

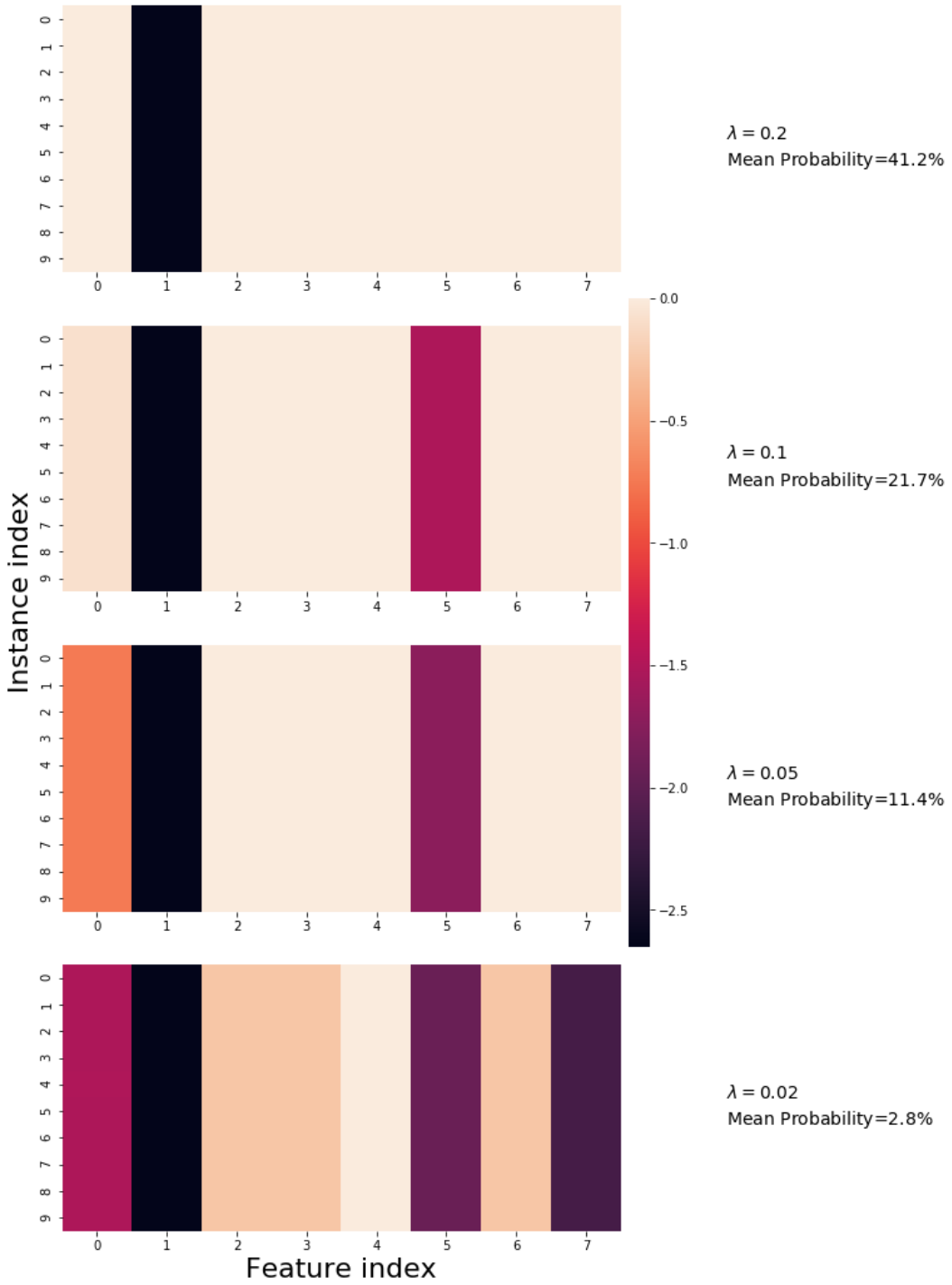


Figure 15: Visualization of learned masks for 4 cases with different values λ : $0.2, 0.1, 0.05, 0.02$ (top to bottom). Hyperparameter $\mu = 0.2$ in all cases. Each case also shows mean probability ($term\_1$ in equation 6). Each row represents an instance (we have 10 instances in every case) and each column represents a feature of the learned mask. Setting smaller value of λ requires more features of the mask to have non-zero values (meaning, makes those features discriminative for a target class), however it also reduces mean probability of the target class. Initial mean probability (when all masks are 0) is $\sim 94\%$, so by just changing feature 1 we can reduce mean probability down to 41.2%.

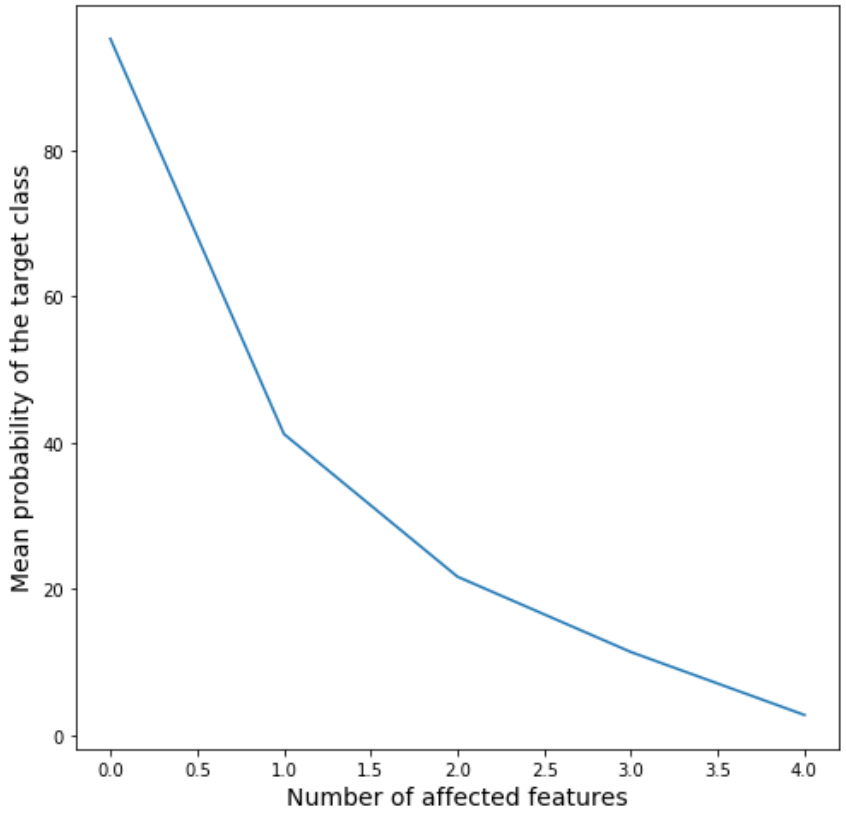


Figure 16: Curve that shows how mean probability of the target class changes with respect to number of affected features (number of non-zero entries in the masks). This plot has been obtained from figure 15 in order to have a better visualization for mean probabilities.

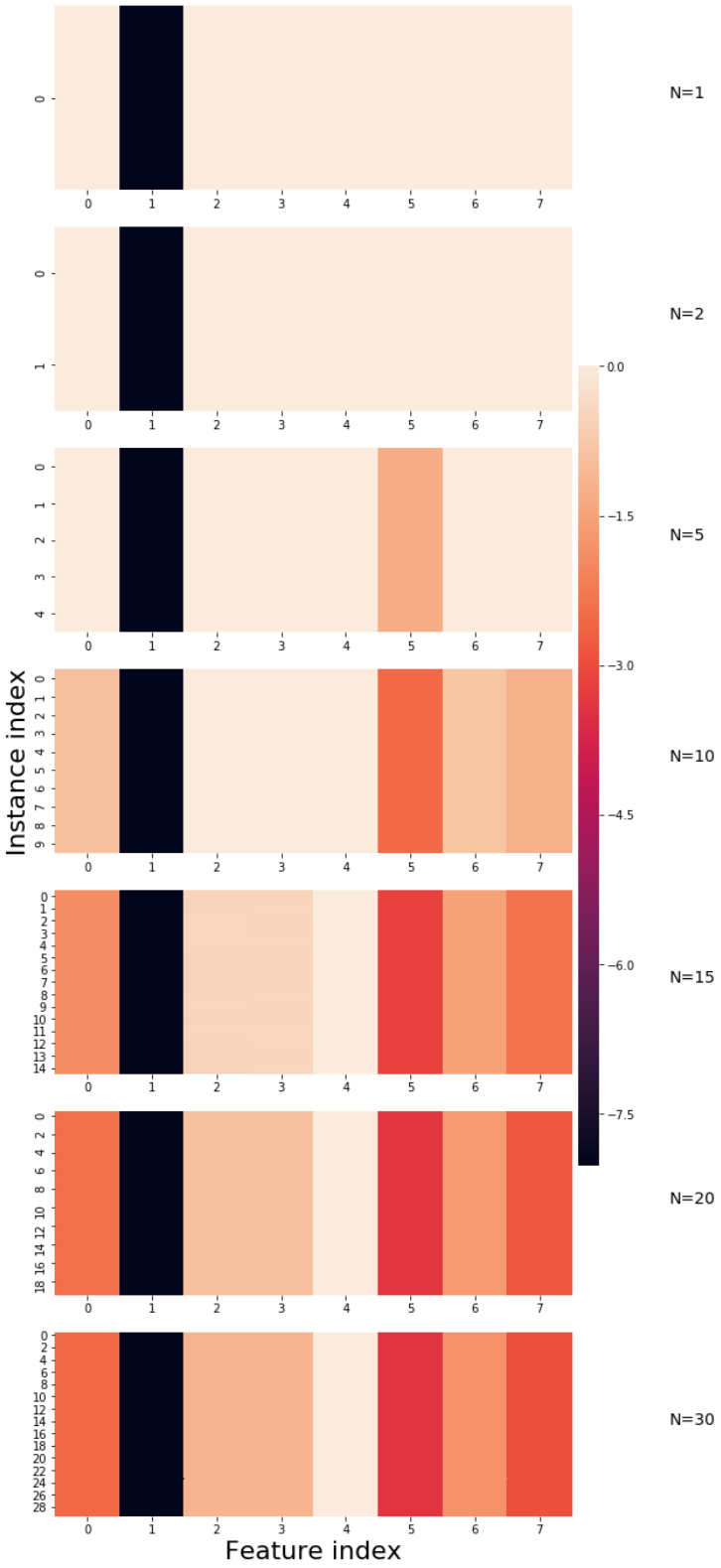


Figure 17: Visualization of learned masks for different number of input data points: $1, 2, 5, 10, 15, 20, 30$. Hyperparameters stay the same $\lambda = 0.02$ and $\mu = 0.2$. Each row represents an instance (we have different instances in every case) and each column represents a feature of the learned mask. Starting from $N = 10 - 15$, the masks start to generalize and new data points do not affect their values, which shows robustness of the framework with respect to the number of data points for a sufficient enough number of data points.

Figure 18: VGGNet [32] configurations: VGG11, VGG13, VGG16.

Figure 19: Aggregated feature maps for 3 classes: 'Tiger cat', 'Lynx' and 'Arctic fox', which are discussed in subsection 4.2.1. Each aggregated feature map is visualized as a heatmap for given image. Every heatmap represents the most discriminative region for a given class.

Figure 20: Aggregated feature maps for 2 classes: ‘Cat’ and ‘Dog’, which are discussed in subsection 4.2.1. Each aggregated feature map is visualized as a heatmap for given image. Every heatmap represents the most discriminative region for a given class.

Figure 21: Example on how proposed method helps with troubleshooting. In this example, VGG16 misclassified image as ‘Egyptian cat’ class. However, after observing the heatmap, we can see that there is a lot of weight in the bottom of the image, where texture indeed resembles the texture of Egyptian cat.

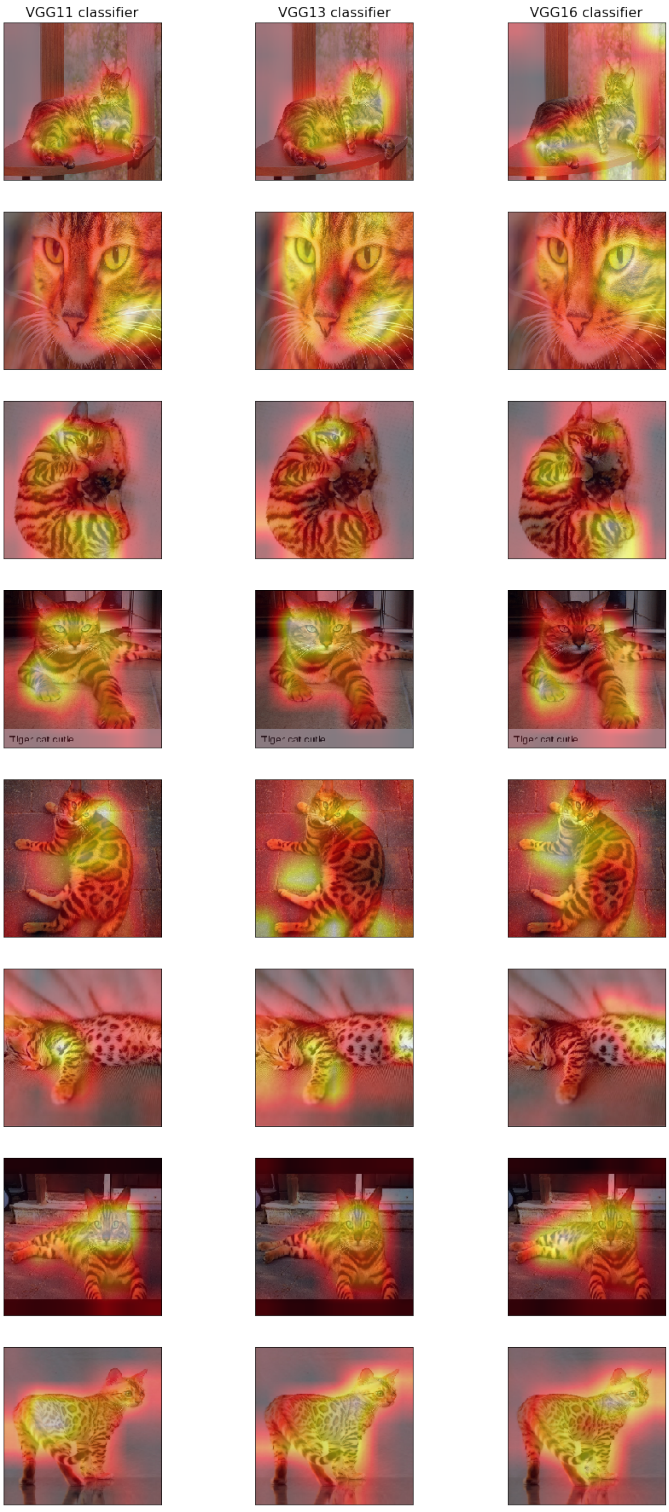


Figure 22: Comparison of 3 different configurations of VGGNet for explaining class 'Tiger cat'. Nevertheless there is a big overlap in explanations for different classifiers, we can still see the differences. For example, on rows 3 and 4 VGG16 does not take into account cat's face, while VGG11 and VGG13 do.

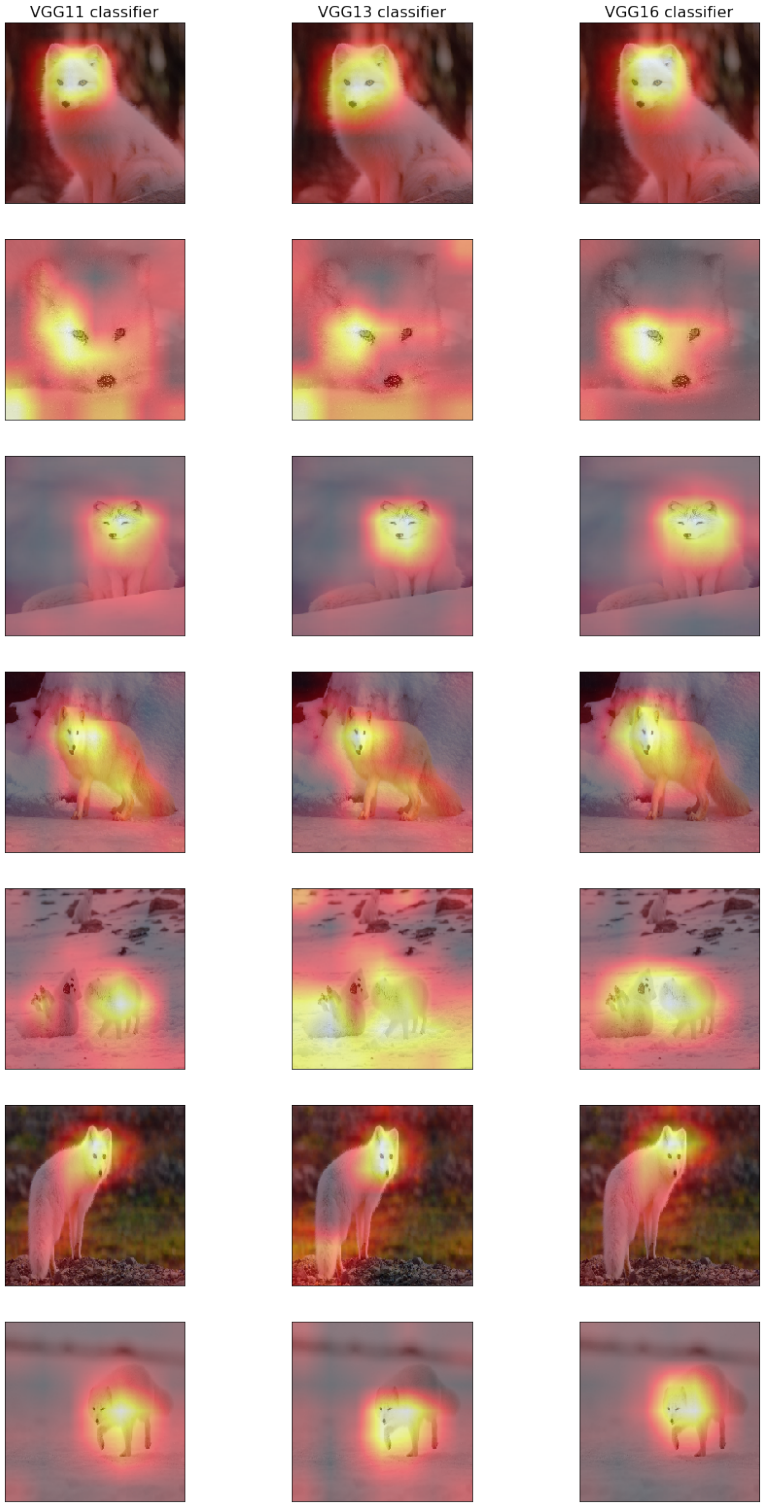


Figure 23: Comparison of 3 different configurations of VGGNet for explaining class 'Arctic fox'. Nevertheless there is a big overlap in explanations for different classifiers, we can still see the differences. For example, on row 2 VGG11 and VGG13 take into account snow, but VGG16 does not. Another big difference is on row 5, where VGG11 almost does not take into account left fox, VGG13 takes into account a lot of snow, while VGG16 only looks at 2 foxes.

```
From: dpage@ra.csc.ti.com (Doug Page)
Subject: Re: Sr-71 in propoganda films?
Nntp-Posting-Host: ra
Organization: Texas Instruments
Distribution: sci
Lines: 28

In article <1993Apr5.220610.1532@sequent.com>, bigfoot@sequent.com (Gregory Smith) writes:
|> mccall@mksol.dseg.ti.com (fred j mccall 575-3539) writes:
|>
|> >In <1phv98$jbk@access.digex.net> prb@access.digex.com (Pat) writes:
|>
|>
|> >>THe SR-71 stopped being a real secret by the mid 70's.
|> >>I had a friend in high school who had a poster with it's picture.
|>
|> >It was known well before that.  I built a model of it sometime in the
|> >mid 60's, billed as YF-12A/SR-71.  The model was based on YF-12A specs
|> >and had a big radar in the nose and 8 AAMs in closed bays on the
|> >underside of the fuselage.  The description, even then, read "speeds
|> >in excess of Mach 3 at altitudes exceeding 80,000 feet."
|>
|> L.B.J. publically announced the existance of the Blackbird program
|> in 1964.


He's also the one who dubbed it the SR-71 - it was the RS-71 until LBJ
mippselled (sic) it.

FWIW,

Doug Page

***  The opinions are mine (maybe), and don't necessarily represent those  ***
***  of my employer.                                                       ***
```

Figure 24: Sample document from *20 NewsGroups* dataset, which belongs to the 'sci.space' category.

5 Conclusions

This thesis starts from discussion about issues in black-box systems and why transparency is very important in different applications. We then define notion of *explanation* and *explainer*, and discuss various aspect of interpretability. After that we provide an overview of globally interpetable models and discuss various local explainers.

In the section 3 we propose a new framework for explaining machine learning classifiers. It is based on the ideas of *local* explainers such as counterfactual explainers, but provides a *global* perspective on the target class. To achieve *global* perspective, we are finding set of counterfactuals for given data points, such that counterfactuals obtained by almost the same semantically meaningful perturbations. We propose a formal definition of the framework, which is model-agnostic and only requires the output of the model to be differentiable with respect to the input of the model. Then, we specify concrete settings for tabular, image and text data.

In the section 4 we conduct three sets of experiments for tabular, image and text data. For tabular data we analyze role of hyperparameters λ and μ, and show that $\lambda > 0$ and $\mu > 0$ setting leads to consistent explanations (see figure 13). Then sensitivity of hyperparameter λ and robustness of the framework with respect to the number of data points are analyzed. For image data we provide explanations by computing aggregated maps, upsampling those and plotting them as a heatmap on input image. Based on this visualization method, we compare different classifiers of VGGNet family. For text data we provide explanations for classes by displaying top words. We show consistency of explanations for the setting $\lambda > 0$ and $\mu > 0$ and explain how proposed framework can be utilized for model troubleshooting.

In our opinion, the main weakness of the proposed framework is in the need to specify the mask with semantically meaningful entries. Even though we propose concrete settings for three types of data, it might not be trivial to specify the mask, which will provide helpful explanations for the given problem.

There are number of different directions for future work that we would like to explore. First, we would like to explore applications of our framework to other types of data, including video and speech. Second, we would like get rid of the main weakness of our framework and find a way to specify the mask to any problem, regardless of the problem. Finally, we would like to do more theoretical work on guarantees that obtained explanation is an explanation for the target class, not just by showing it empirically. Regarding this issue, we would like to explore methods to sample data points for which framework finds the counterfactuals, such that sampled data points are diverse enough and can serve as good representatives for the target class.

References

[1] URL http://archive.ics.uci.edu/ml/datasets/twenty+newsgroups.

[2] URL https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing.

[3] URL https://github.com/destinityx2/ml-explain/tree/master/experiments.

[4] URL https://archive.ics.uci.edu/ml/datasets/diabetes.

[5] Introduction to model builder scorecard. url: https://www.fico.com/en/latest-thinking/white-papers/introduction-to-model-builder-scorecard. accessed on 04.03.2019. 2011. URL https://www.fico.com/en/latest-thinking/white-papers/introduction-to-model-builder-scorecard.

[6] Nahla Barakat and Andrew P. Bradley. Rule extraction from support vector machines: A review. *Neurocomputing*, 74(1):178 – 190, 2010. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2010.02.016. URL http://www.sciencedirect.com/science/article/pii/S0925231210001591. Artificial Brains.

[7] Aylin Caliskan, Joanna J. Bryson, and Arvind Narayanan. Semantics derived automatically from language corpora contain human-like biases. *Science*, 356 (6334):183–186, 2017. ISSN 0036-8075. doi: 10.1126/science.aal4230. URL http://science.sciencemag.org/content/356/6334/183.

[8] Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1721–1730, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3664-2. doi: 10.1145/2783258.2788613. URL http://doi.acm.org/10.1145/2783258.2788613.

[9] Chun-Hao Chang, Elliot Creager, Anna Goldenberg, and David Duvenaud. Explaining image classifiers by counterfactual generation, 2018.

[10] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995. ISSN 0885-6125. doi: 10.1023/A:1022627411411. URL https://doi.org/10.1023/A:1022627411411.

[11] Mark W. Craven and Jude W. Shavlik. Extracting tree-structured representations of trained networks. In *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS'95, pages 24–30, Cambridge, MA, USA, 1995. MIT Press. URL http://dl.acm.org/citation.cfm?id=2998828.2998832.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[13] Dumitru Erhan, Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. 2009.

[14] R. C. Fong and A. Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 3449–3457, Oct 2017. doi: 10.1109/ICCV.2017.371.

[15] Ruth C. Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. doi: 10.1109/iccv.2017.371. URL http://dx.doi.org/10.1109/ICCV.2017.371.

[16] Alex A. Freitas. Comprehensible classification models: A position paper. *SIGKDD Explor. Newsl.*, 15(1):1–10, March 2014. ISSN 1931-0145. doi: 10.1145/2594473.2594475. URL http://doi.acm.org/10.1145/2594473.2594475.

[17] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Computing Surveys*, 51(5):1–42, Aug 2018. ISSN 0360-0300. doi: 10.1145/3236009. URL http://dx.doi.org/10.1145/3236009.

[18] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *Signal Processing Magazine*, 2012.

[19] B. Brown J. Bughin R. Dobbs C. Roxburgh J. Manyika, M. Chui and A. Byers. Big data: The next frontier for innovation, competition, and productivity. *McKinsey Global Institute*, 2011.

[20] U. Johansson and L. Niklasson. Evolving decision trees using oracle guides. In *2009 IEEE Symposium on Computational Intelligence and Data Mining*, pages 238–244, March 2009. doi: 10.1109/CIDM.2009.4938655.

[21] R. Krishnan, G. Sivakumar, and P. Bhattacharya. Extracting decision trees from trained neural networks. *Pattern Recognition*, 32(12):1999 – 2009, 1999. ISSN 0031-3203. doi: https://doi.org/10.1016/S0031-3203(98)00181-2. URL http://www.sciencedirect.com/science/article/pii/S0031320398001812.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999134.2999257.

[23] Yann Lecun, Koray Kavukcuoglu, and Clement Farabet. Convolutional networks and applications in vision. pages 253–256, 05 2010. doi: 10.1109/ISCAS.2010.5537907.

[24] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. *Lecture Notes in Computer Science*, page 740–755, 2014. ISSN 1611-3349. doi: 10.1007/978-3-319-10602-1_48. URL http://dx.doi.org/10.1007/978-3-319-10602-1_48.

[25] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[26] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, Feb 2019. ISSN 0004-3702. doi: 10.1016/j.artint.2018.07.007. URL http://dx.doi.org/10.1016/j.artint.2018.07.007.

[27] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[28] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. *CoRR*, abs/1602.04938, 2016. URL http://arxiv.org/abs/1602.04938.

[29] Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, Nov 1987. ISSN 1573-0565. doi: 10.1007/BF00058680. URL https://doi.org/10.1007/BF00058680.

[30] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[31] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. doi: 10.1109/iccv.2017.74. URL http://dx.doi.org/10.1109/ICCV.2017.74.

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[33] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2013.

[34] James W. Smith, James E. Everhart, William C. Dickson, William C Knowler, and Richard S. Johannes. Using the adap learning algorithm to forecast the onset of diabetes mellitus. 1988.

[35] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2013.

[36] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015. URL http://arxiv.org/abs/1409.4842.

[37] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016. doi: 10.1109/cvpr.2016.308. URL http://dx.doi.org/10.1109/CVPR.2016.308.

[38] Sandra Wachter, Brent Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the gdpr. *SSRN Electronic Journal*, 2017. ISSN 1556-5068. doi: 10.2139/ssrn.3063289. URL http://dx.doi.org/10.2139/ssrn.3063289.

[39] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9(4):611–629, Aug 2018. ISSN 1869-4101. doi: 10.1007/s13244-018-0639-9. URL https://doi.org/10.1007/s13244-018-0639-9.

[40] M. D. Zeiler, G. W. Taylor, and R. Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *2011 International Conference on Computer Vision*, pages 2018–2025, Nov 2011. doi: 10.1109/ICCV.2011.6126474.

[41] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *ArXiv*, abs/1311.2901, 2013.

[42] Zizhao Zhang, Fuyong Xing, Hai Su, Xiaoshuang Shi, and Lin Yang. Recent advances in the applications of convolutional neural networks to medical image contour detection, 2017.